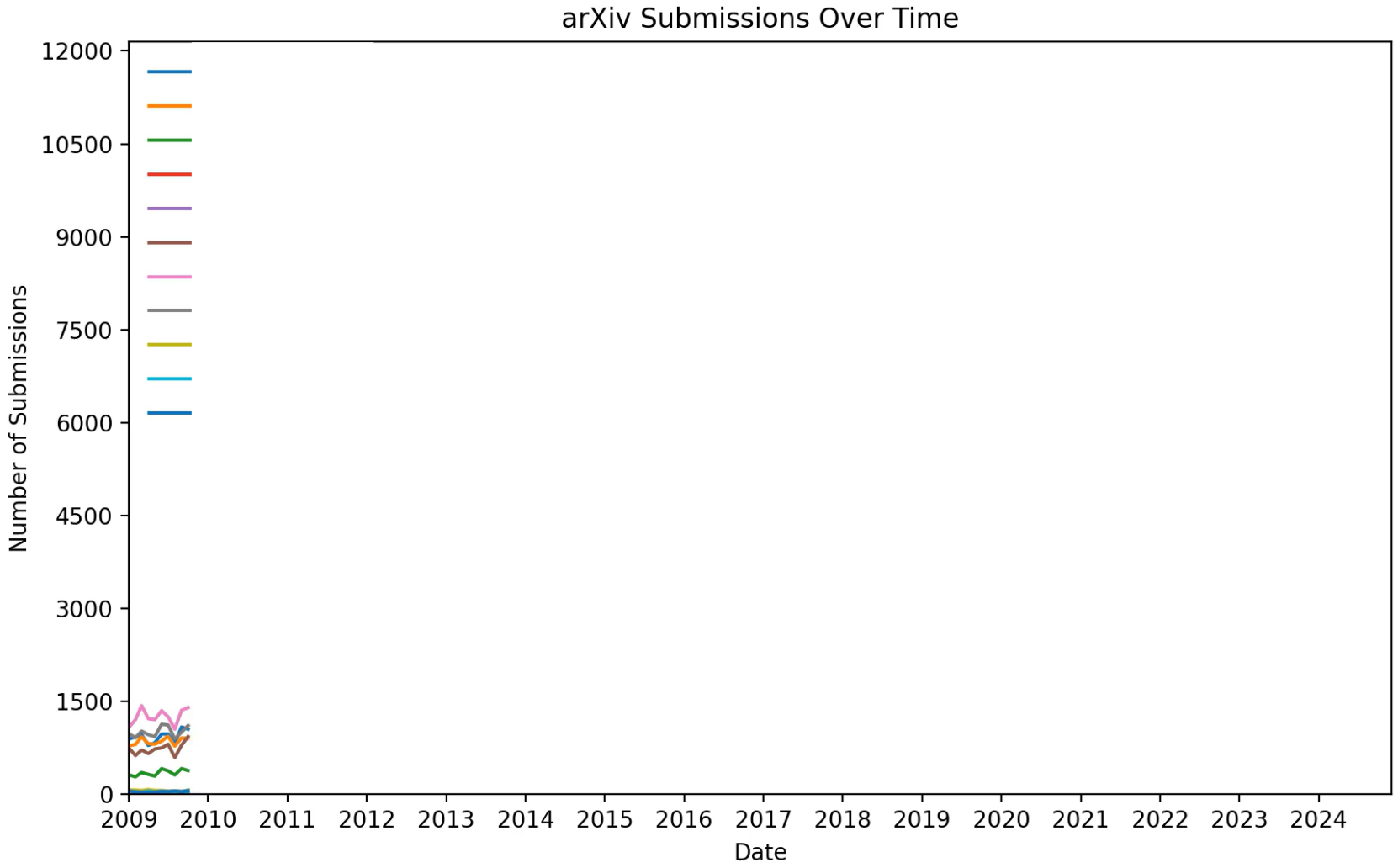
New developments in machine (deep) learning

A practical view

Malte Algren

Which color matches the CS?

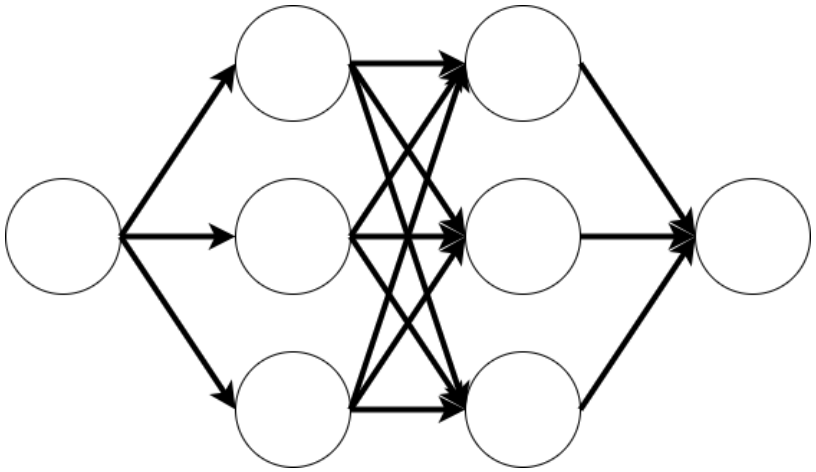
```
['eess',
 'q-fin',
 'econ',
 'hep',
 'astro-ph',
 'cond-mat',
 'stat',
 'q-bio',
 'physics',
 'math',
 'cs']
```

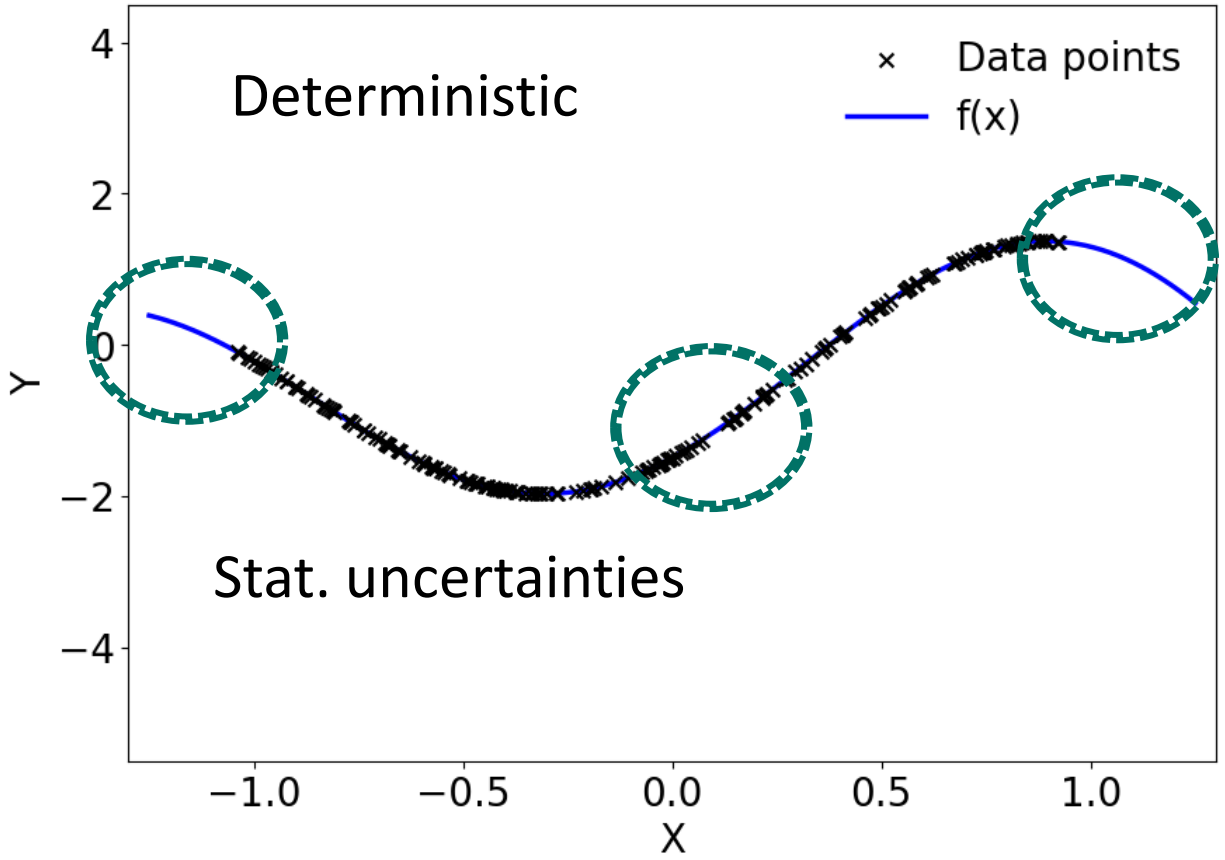


Systematic uncertainties

How to regress $p(y|x)$?

1. Estimating y directly with $f(x)$ (MSE, MAE)
   - You will be estimating $\bar{p}(y|f(x)) = \mu$

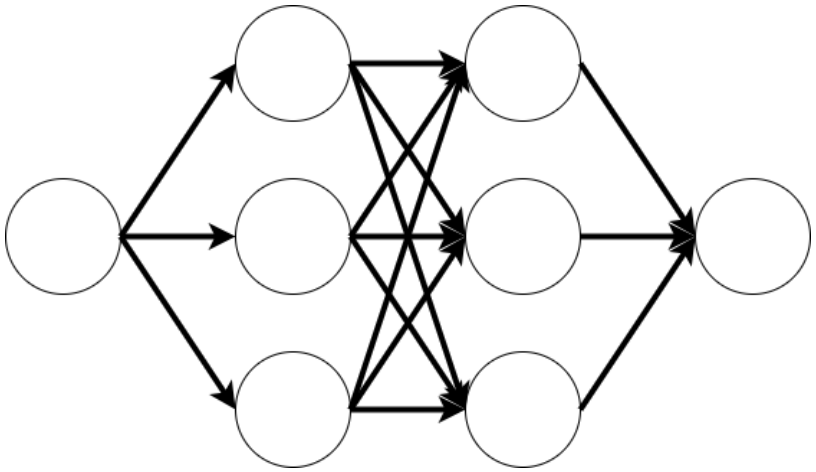


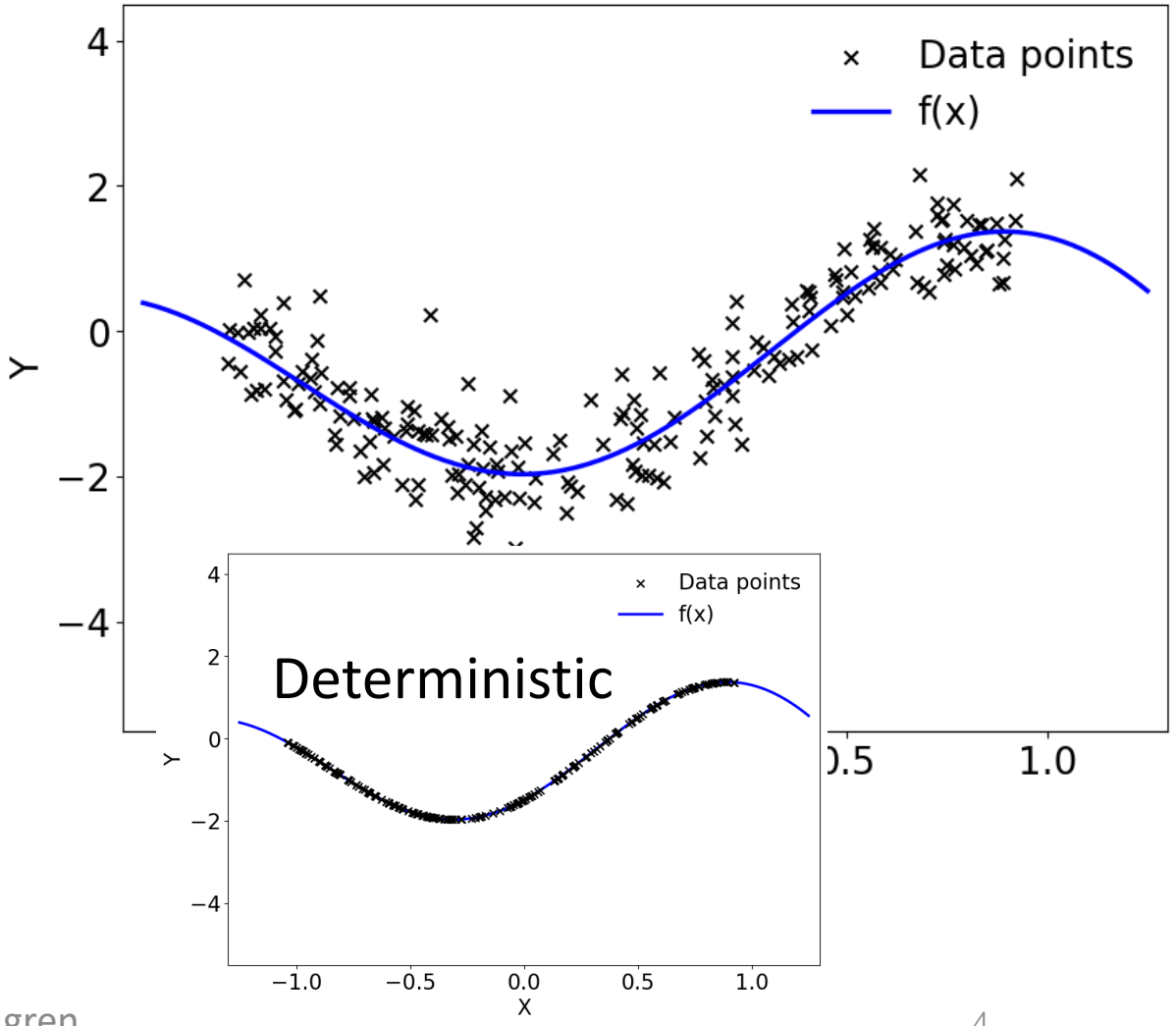


Systematic uncertainties

How to regress $p(y|x)$?

1. Estimating y directly with $f(x)$ (MSE, MAE)
   - You will be estimating $\bar{p}(y|f(x)) = \mu$





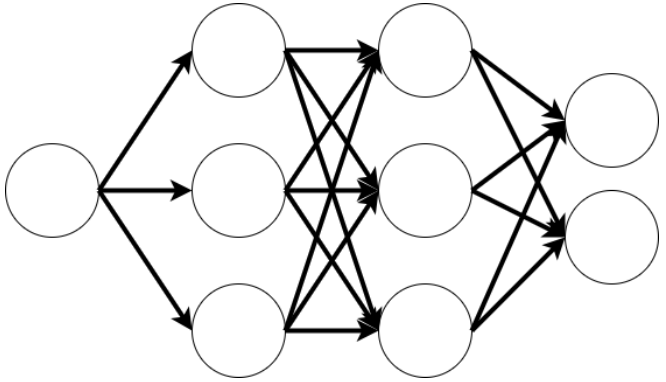
Systematic uncertainties

How to regress $p(y|x)$?

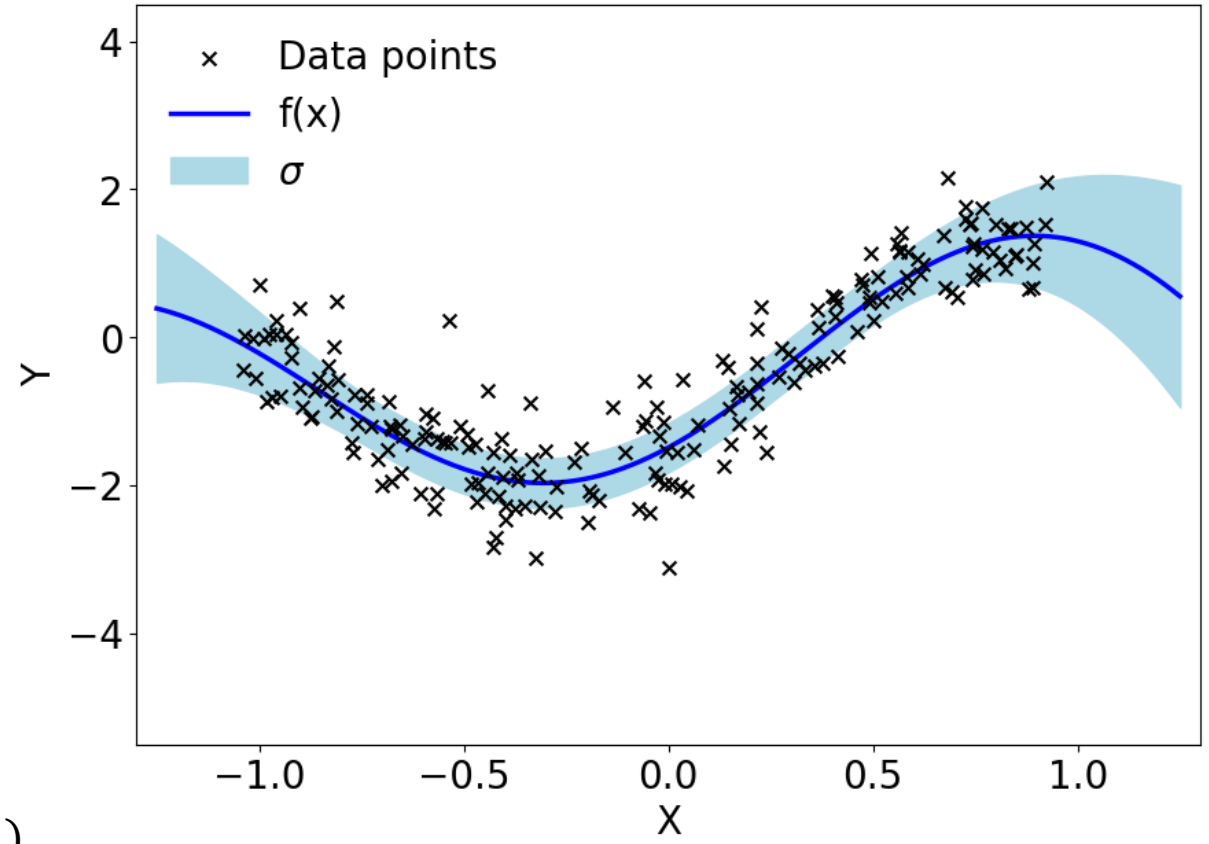
1. Estimating y directly with $f(x)$ (MSE, MAE)

2. Instead of using MSE > log-likelihood

- $p(y|x) = \mathcal{N}(y|f(x)) = \mathcal{N}(y|\mu, \sigma^2)$
- $-\log\big(p(y|x)\big) = \frac{(\mu(x)-y)^2}{2\sigma(x)^2} + \log(\sigma(x))$
  - $\mu(x) \in [-\infty, \infty]$ and $\sigma(x) = (0, \infty]$



$\mathcal{N}(y|\mu, \sigma^2)$

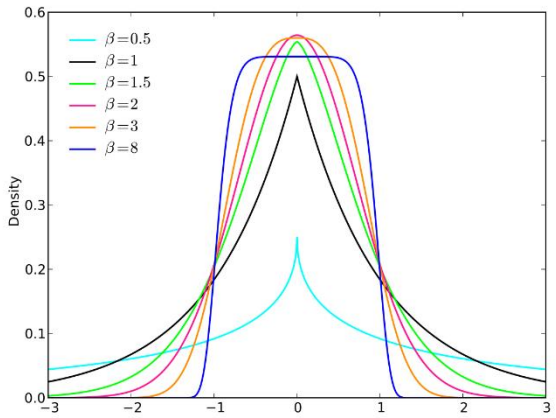


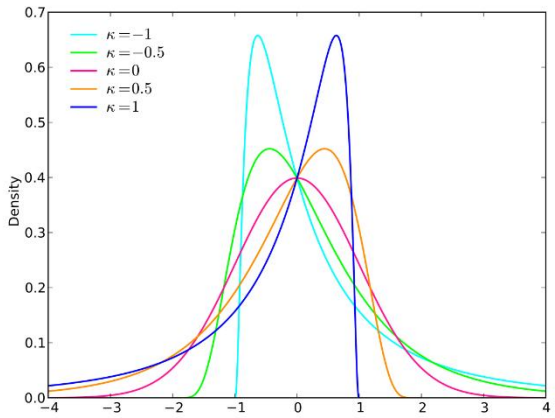
$$\mathcal{L} = \frac{1}{2}\left(\frac{(f(x)-y)^2}{e^u} + \mathrm{u}\right), \qquad \mathrm{u} = \log(\sigma^2)$$

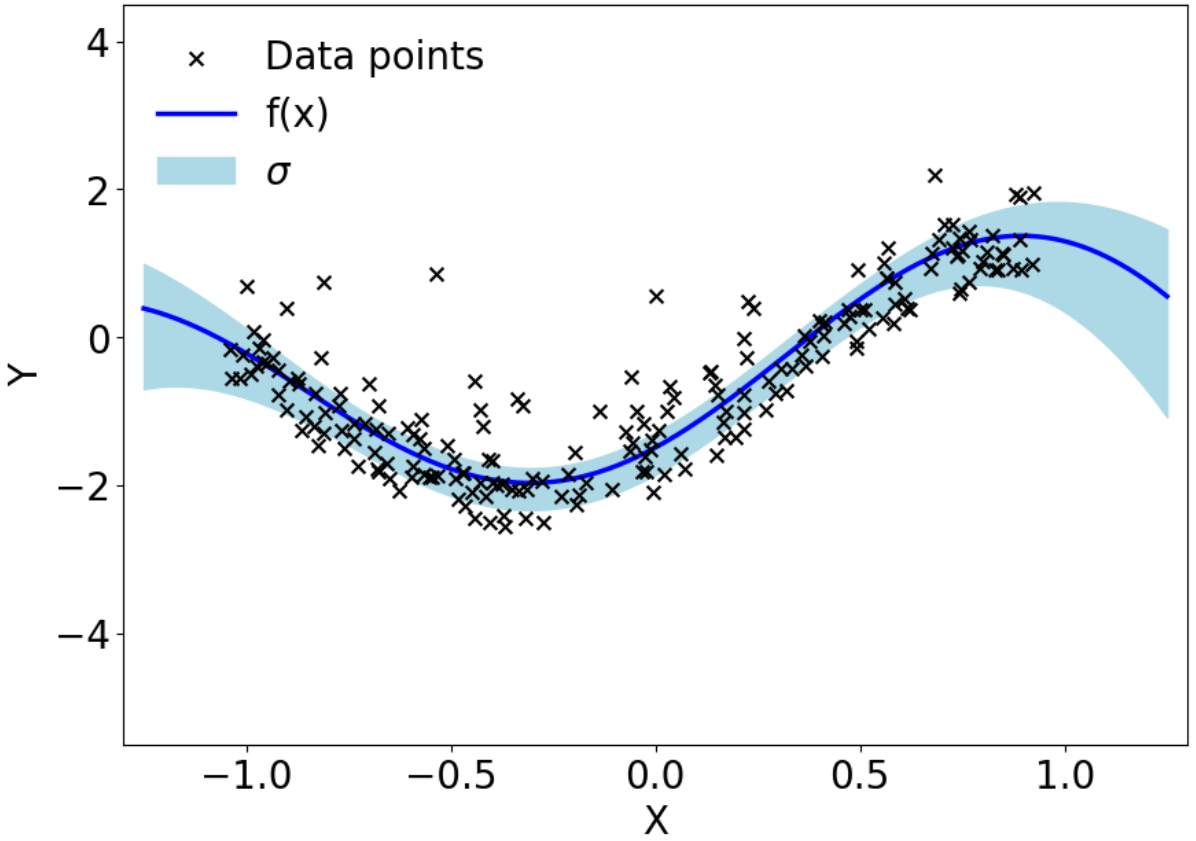
Systematic uncertainties

How to regress $p(y|x)$?

1. Estimating y directly with $f(x)$ (MSE, MAE)
2. Instead of using MSE > log-likelihood
3. Non-gaussian uncertainty (Poisson, etc)
    - Generalized normal distribution
    - Gaussian mixture models



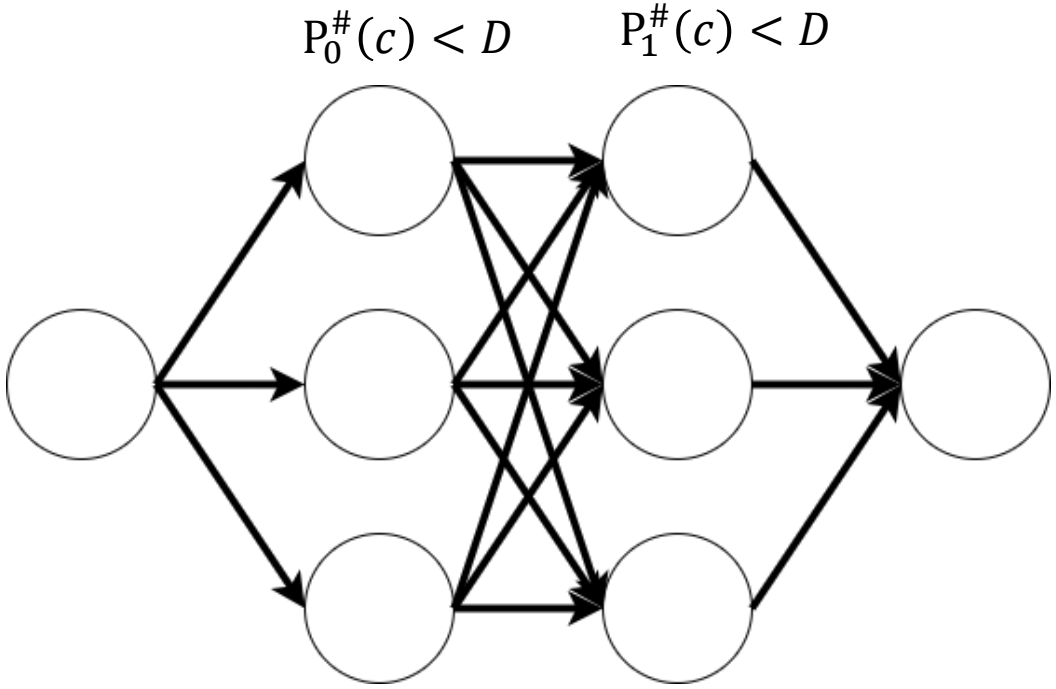


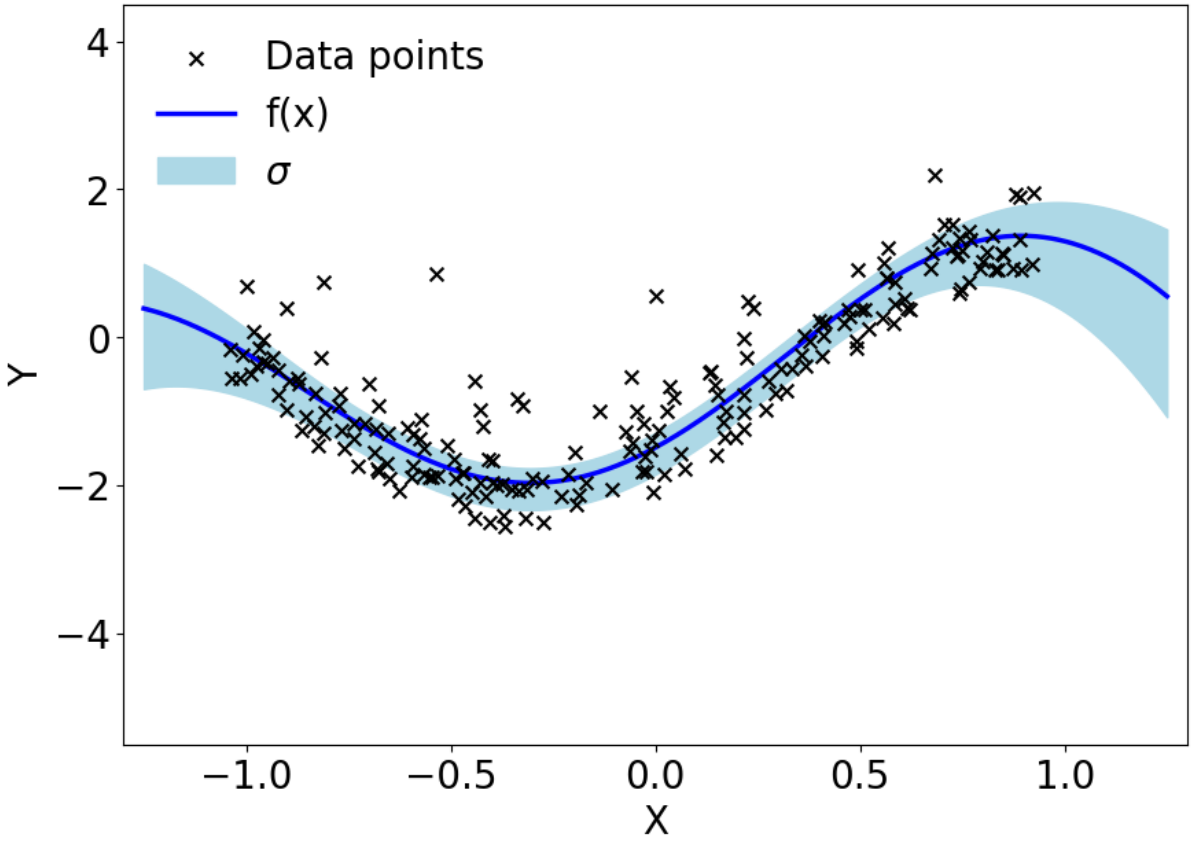


Systematic uncertainties

How to regress $p(y|x)$?

1. Drop-out – drop with a probability *D*

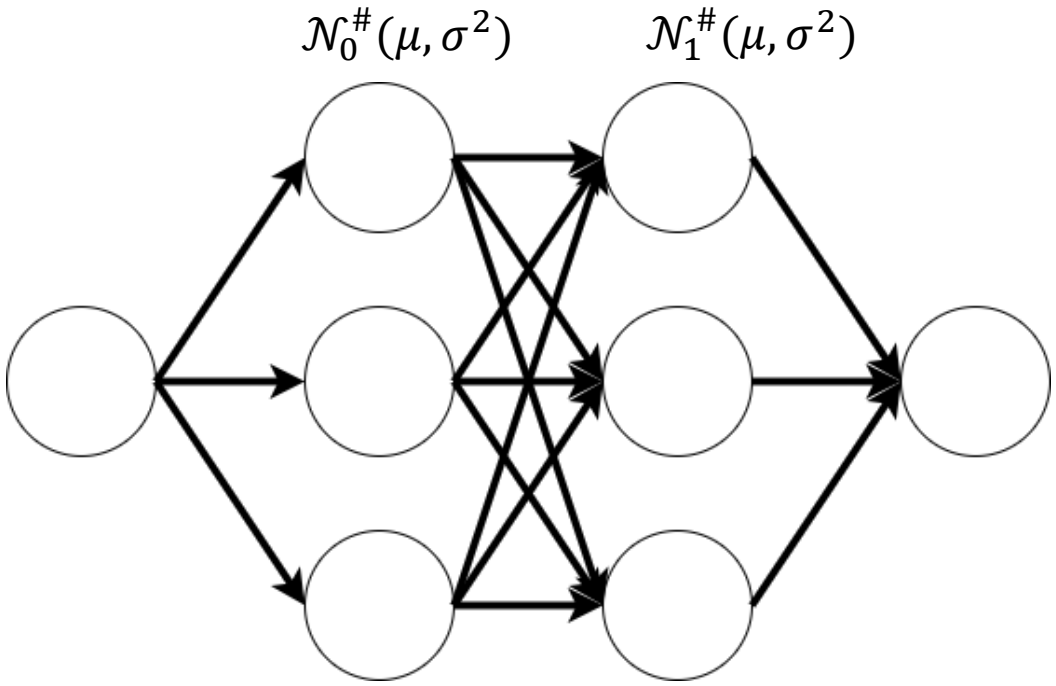


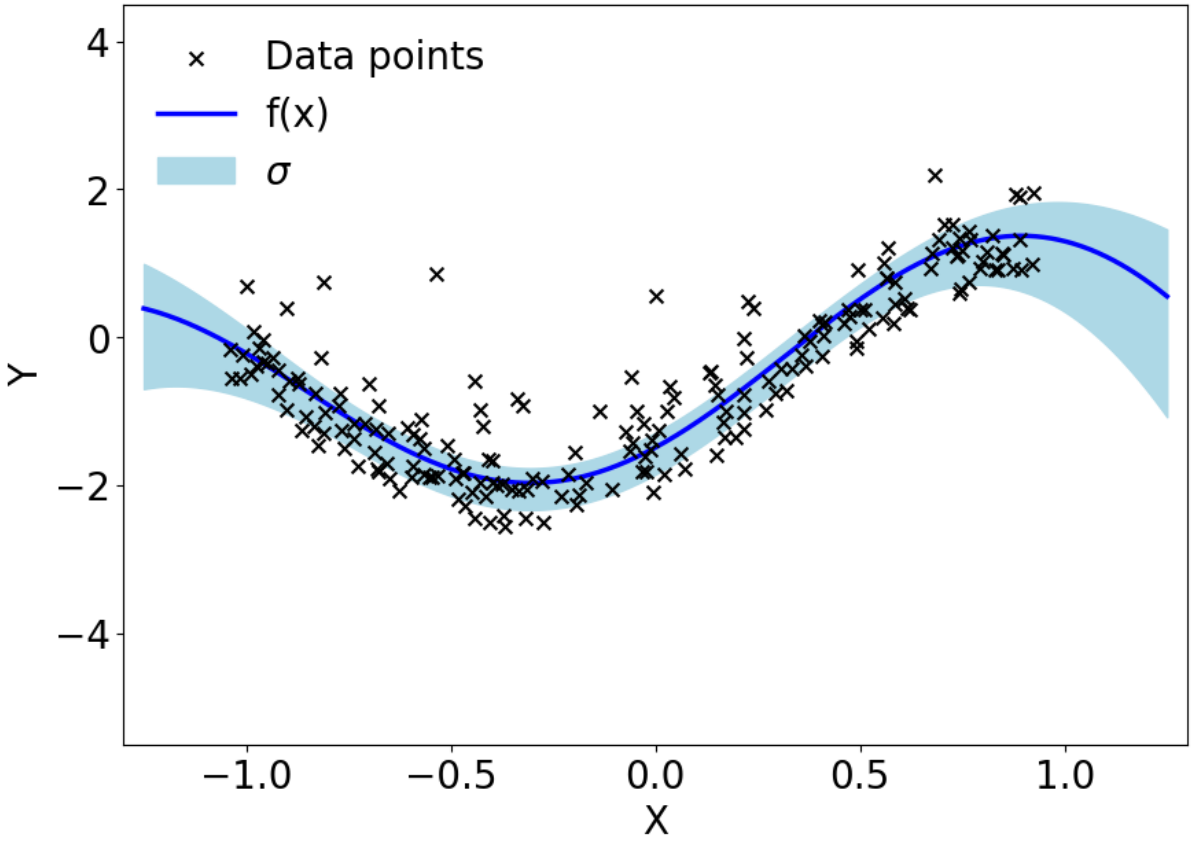


Systematic uncertainties

How to regress $p(y|x)$?

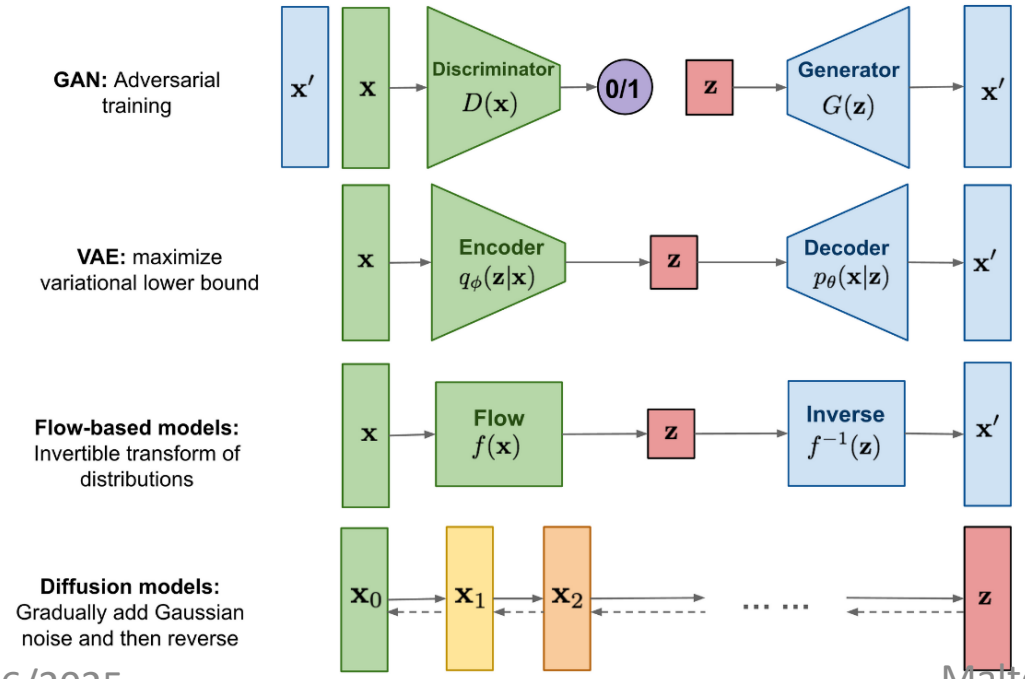
1. Drop-out
2. Bayesian neural network - $w_i^{\#} = \mathcal{N}(\mu, \sigma^2)$

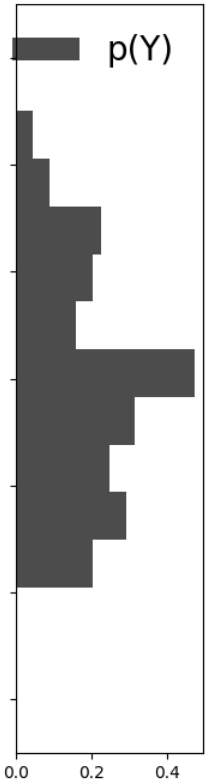




Generative models

- Use generative model to model $p(y)$ or $p(x)$
  - Often, we prefer $p(y|x)$
  - Probabilistic regression / variational inference
  - GANs and VAE would ignore *x*





Generative models

- Very very rarely do we care about $p(y)$
- We (almost) always want $p(y|x)$
  - Example: Image + prompt

Generate this man during his PhD in Paris with a hat

Got the hat right but not the place

Generative models

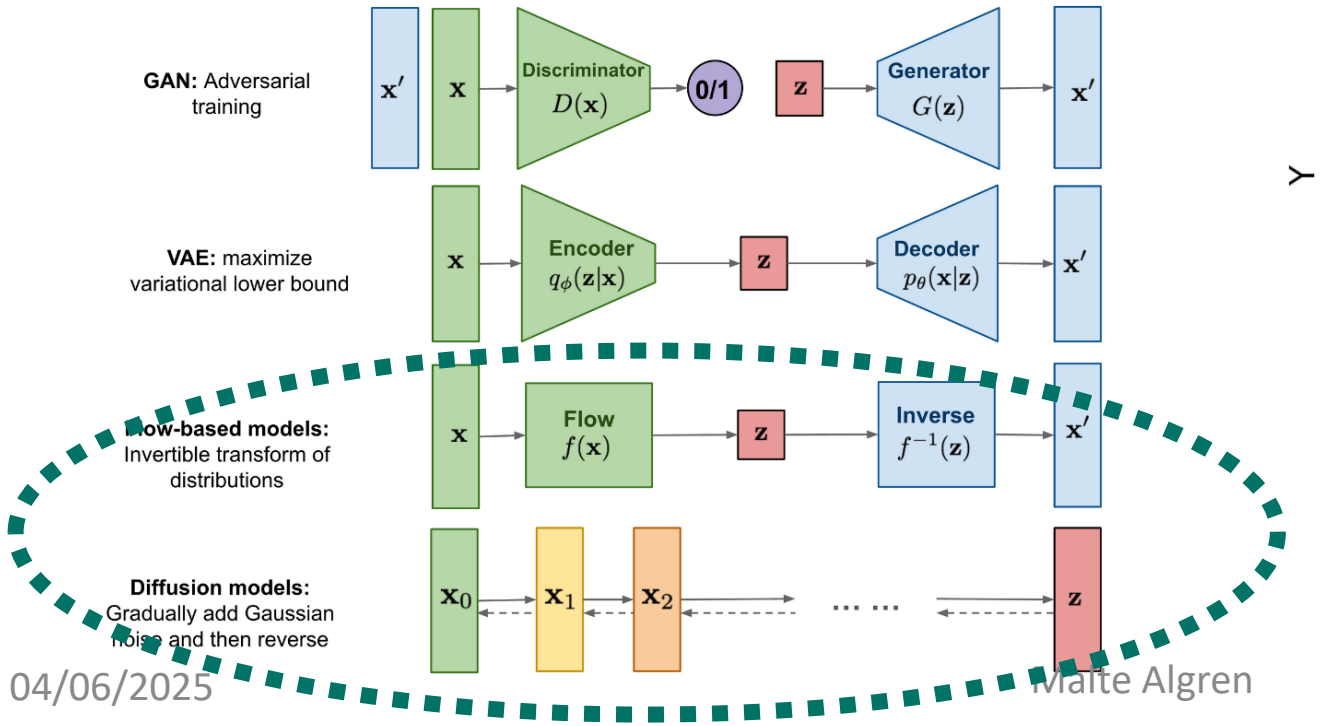
- Very very rarely do we care about $p(y)$
- We (almost) always want $p(y|x)$
  - Example: Image + prompt

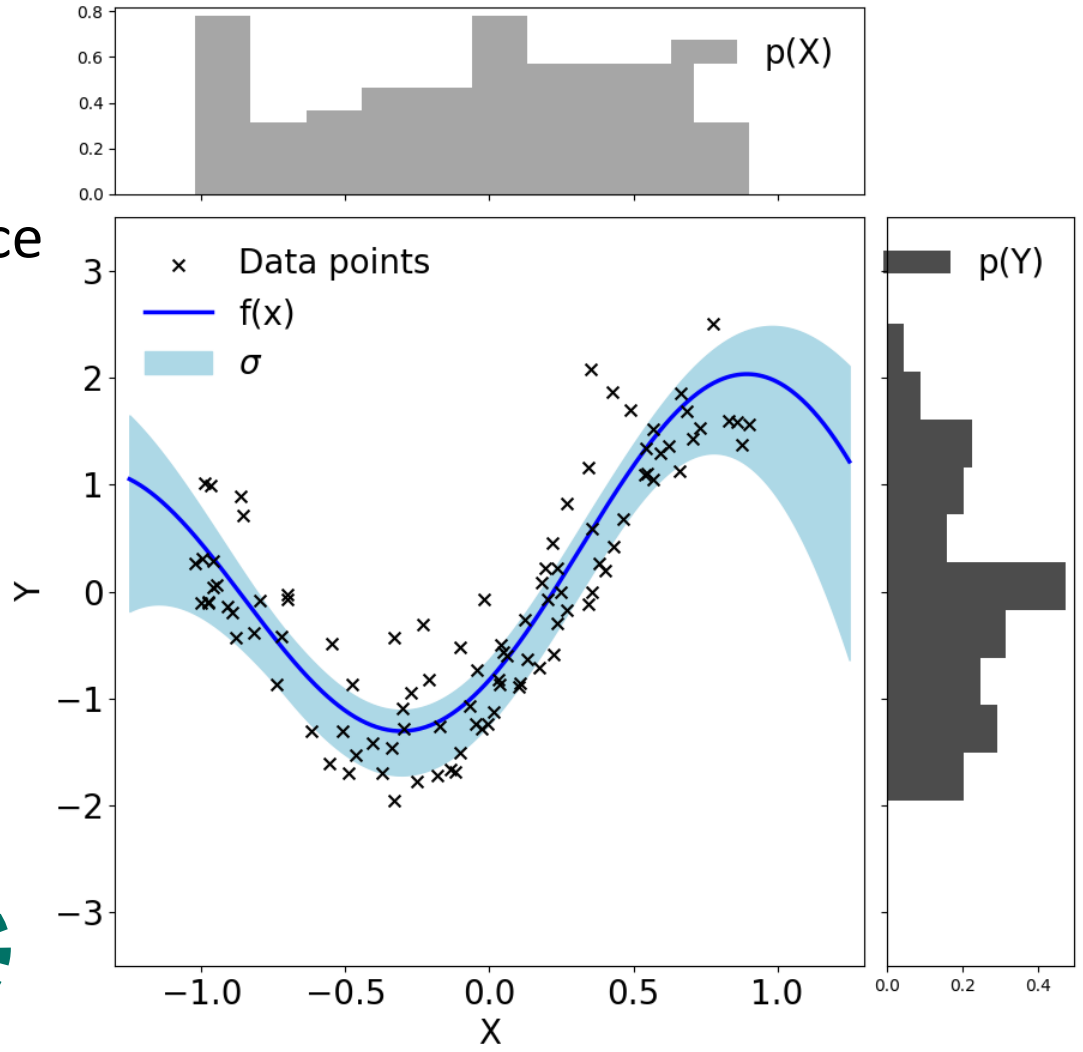
Generate this man during his PhD in Paris with a hat

Correct $p(y|x)$

Generative models

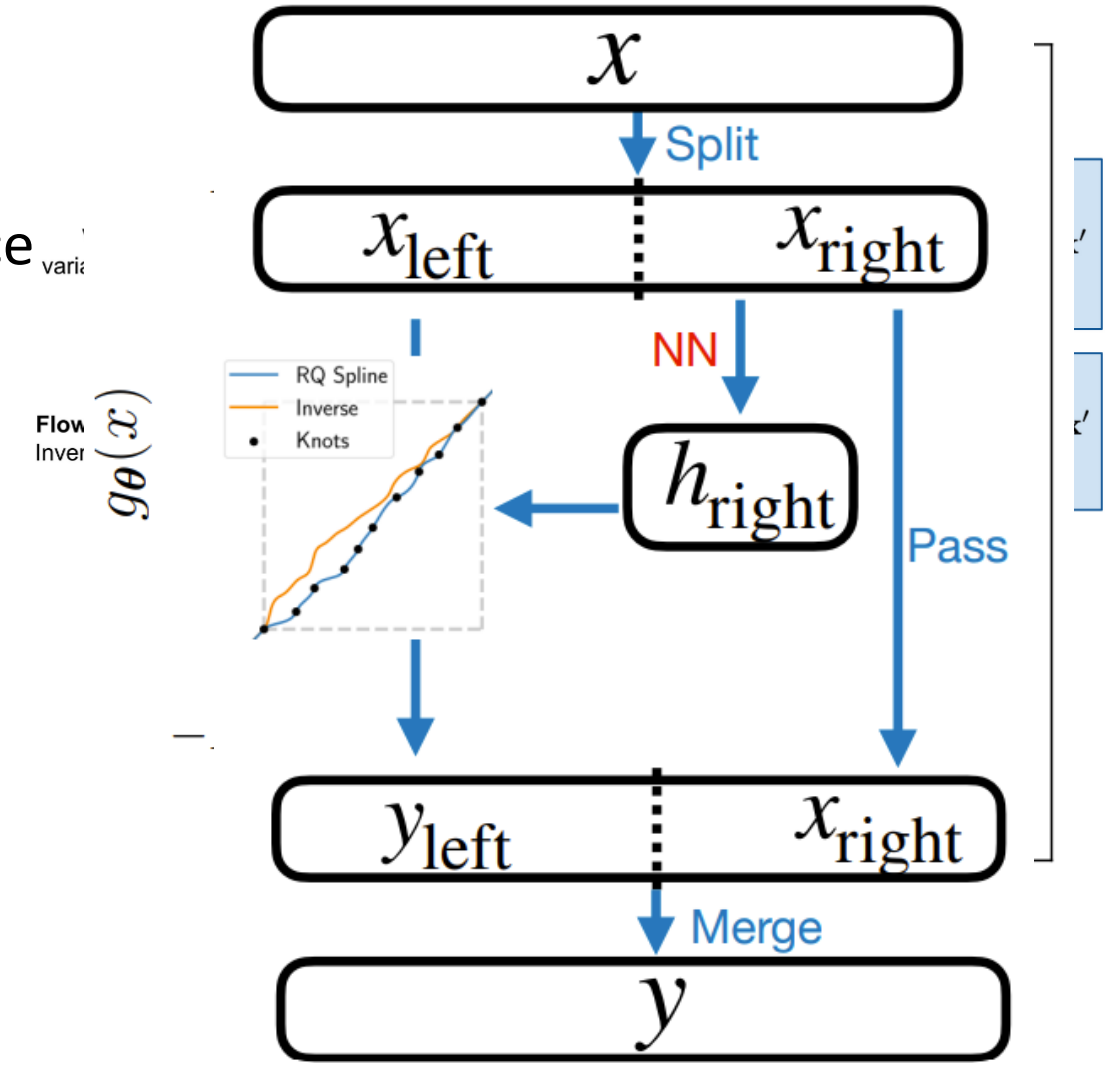
- Use generative model to model $p(y)$ or $p(x)$
  - Often, we prefer $p(y|x)$
  - Probabilistic regression / variational inference
  - GANs and VAE would ignore *x*





Generative models: Invertible neural networks

- Use generative model to model $p(y)$ or $p(x)$
  - Often, we prefer $p(y|x)$
  - Probabilistic regression / variational inference
- Invertible neural networks (Flow):
  - Encoder and decoder is the same network
  - Encoder $f(x) = z$ and decode $f^{-1}(z) = x$
  - Invertible transformations – Splines
  - Condition the spline on subset of *x*
    - $f_\theta(x) = f_{\theta_1} \cdot f_{\theta_2} \cdot \ldots \cdot f_{\theta_{i+1}}(x)$

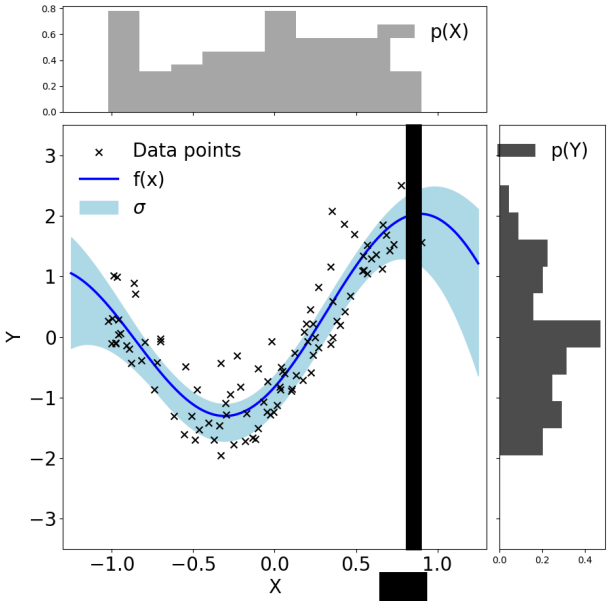


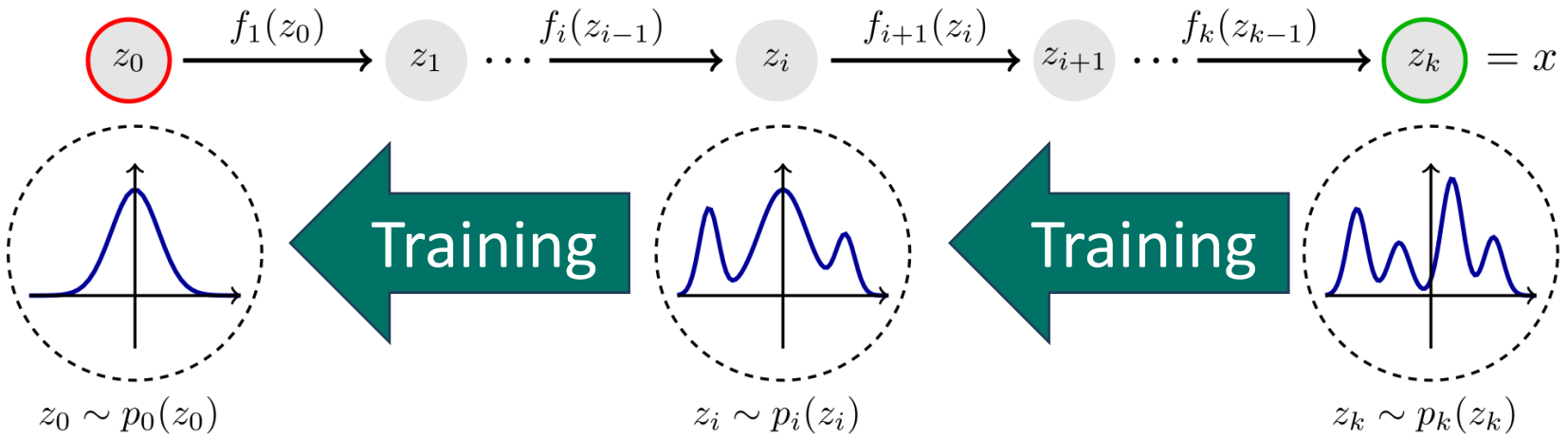
Generative models: Normalising flows

- Trained using change of variable formular (p_0 is tractable)

$$p_1(z_1) = p_0\left(f_1^{-1}(z_1)\right)\left|\det\frac{df_1^{-1}(z_1)}{dz_1}\right|$$

  - $f_\theta(x) = f_{\theta_1} \cdot f_{\theta_2} \cdot \ldots \cdot f_{\theta_{i+1}}(x)$
  - Conditional generation works very well!





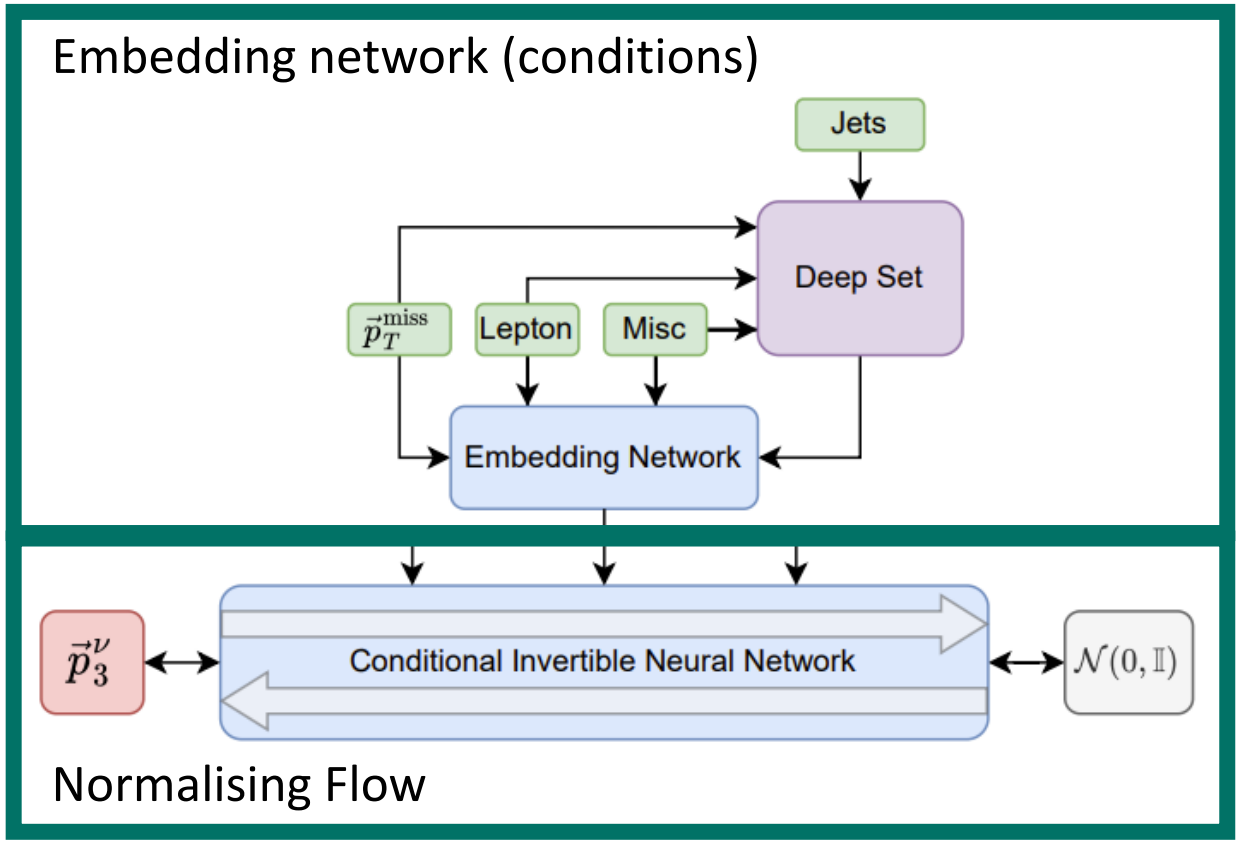
Generative models: Normalising flows

Advantages

- Sample from base to create the posterior
- Relative fast in generation and training
- Stable and regulated
- Well calibrated!

Drawback:

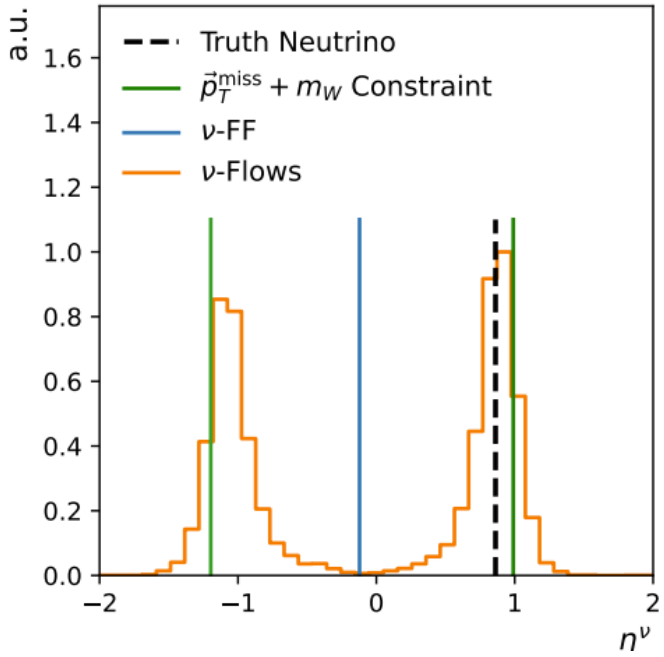
- Works on continuous flat distribution
- More complicated that regular regression

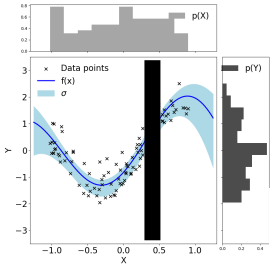


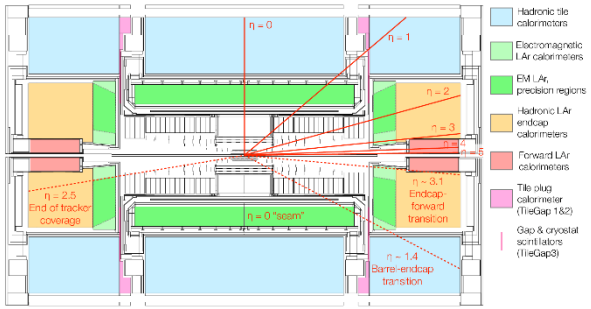
Generative models: Normalising flows

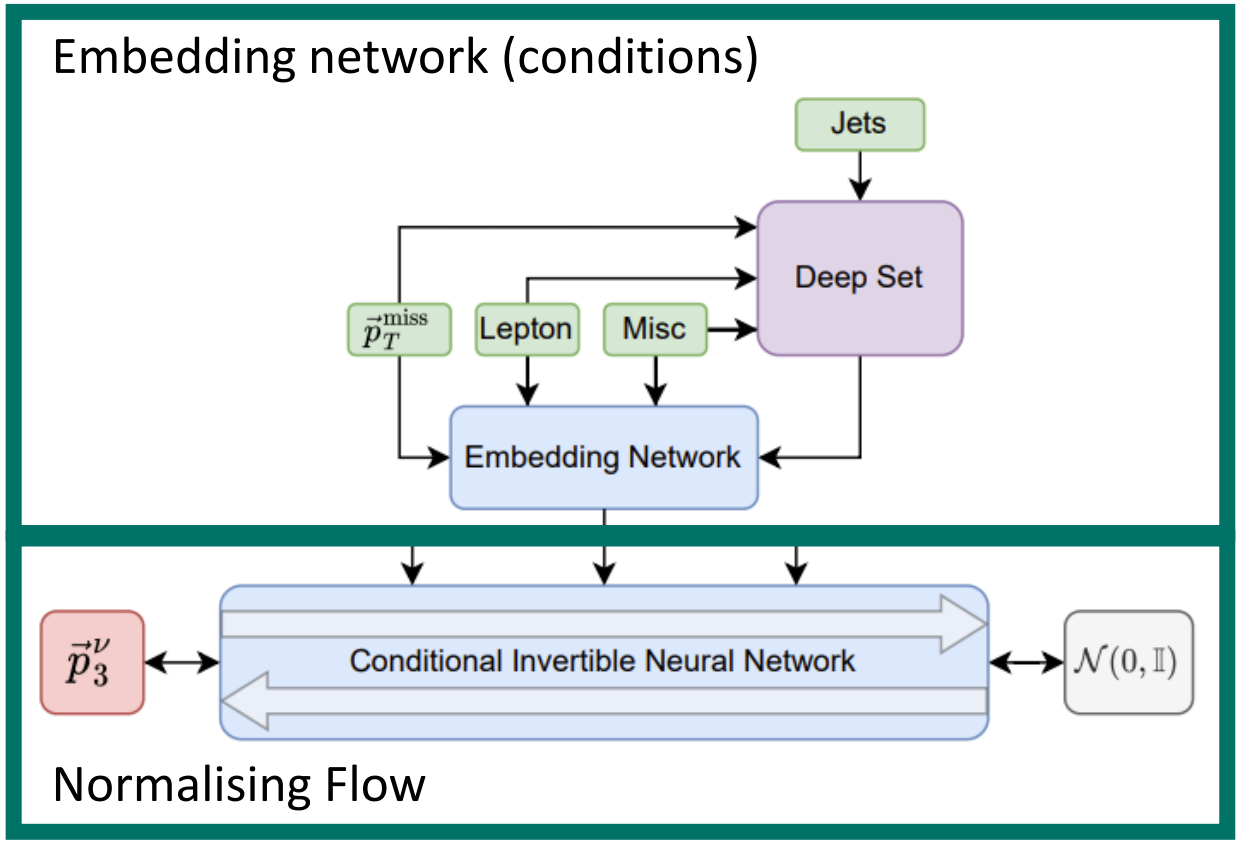
Neutrino reconstruction at LHC

FF: NN regression using *MSE*









How to interpret the output

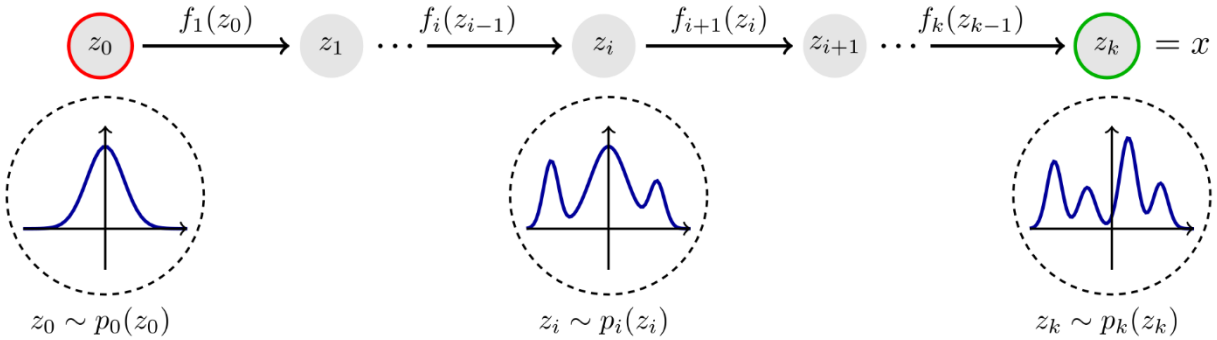
What does the output of a network mean?

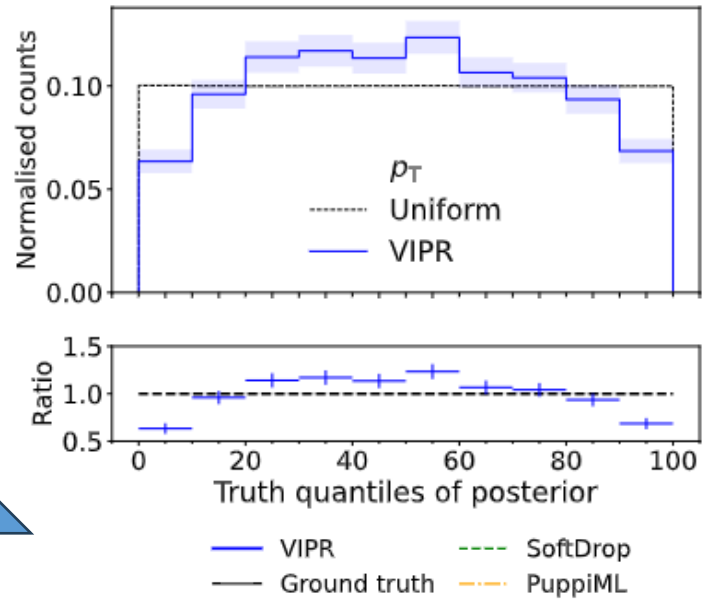
- Classifiers:
  - Predicted probability vs fraction of positives
- Regression (MSE)
  - Only have a single prediction
  - No variance estimate
- Posterior estimation (likelihood, C-generation etc.)
  - Log-likelihood
    - Gaussian distributed pulls
  - Conditional generation (no σ estimate)
    - Uniform distributed truth quantiles

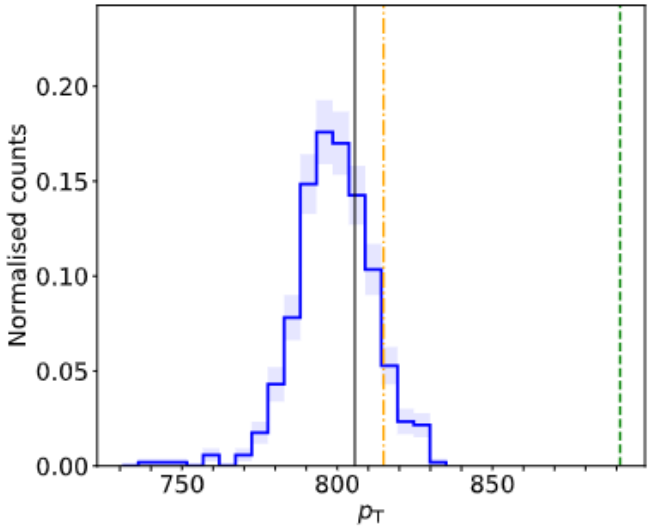
Generative models: Normalising flows

Calibration curve for conditional generative models

1. Generate $p(y_i|x_i)$
2. Measure the quantile of the truth
3. Measure truth quantiles over many posteriors
   - Should follow a uniform distribution



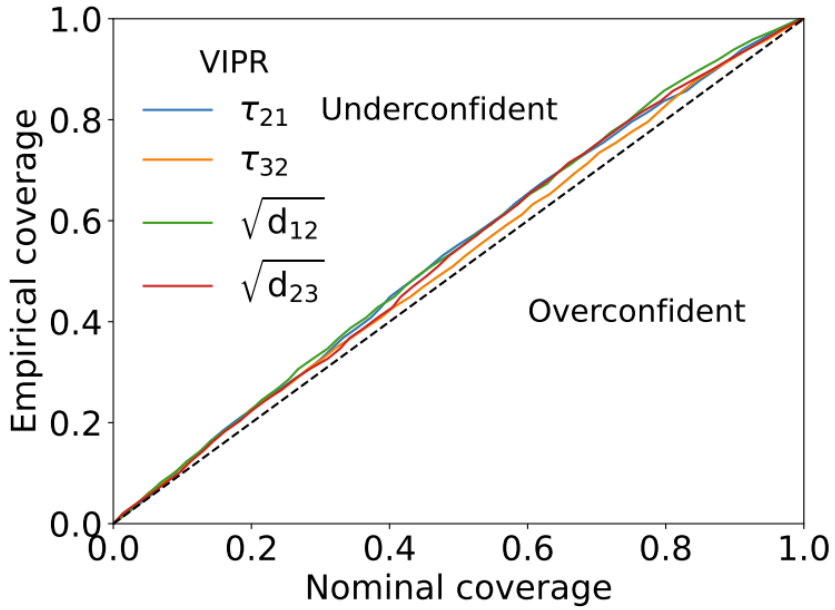


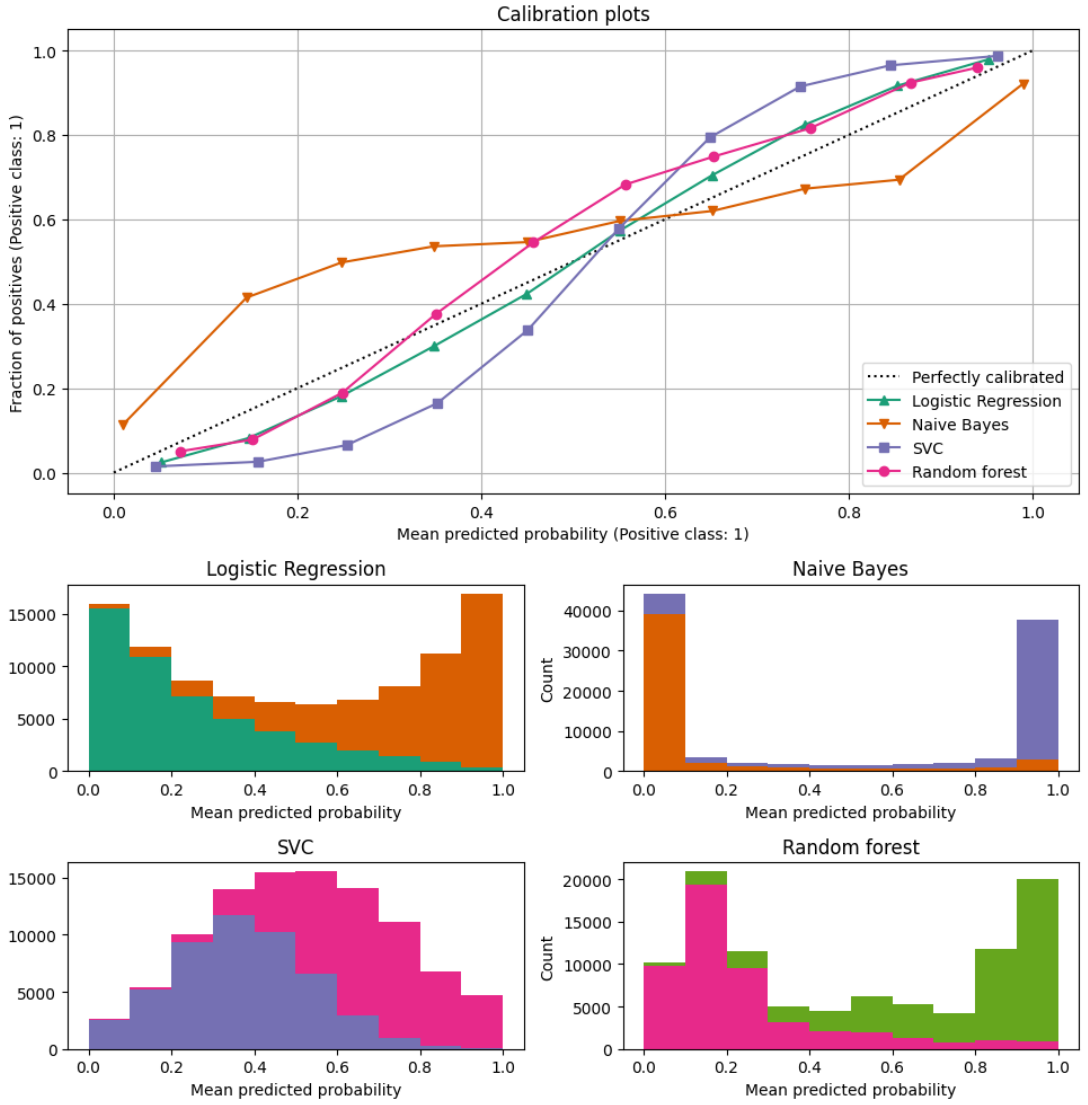


Generative models: Normalising flows

How to ensure that your model is calibrated:

- Classifiers:
  - Predicted probability vs fraction of positives
- Variational inference
  - Estimate the quantile of the truth



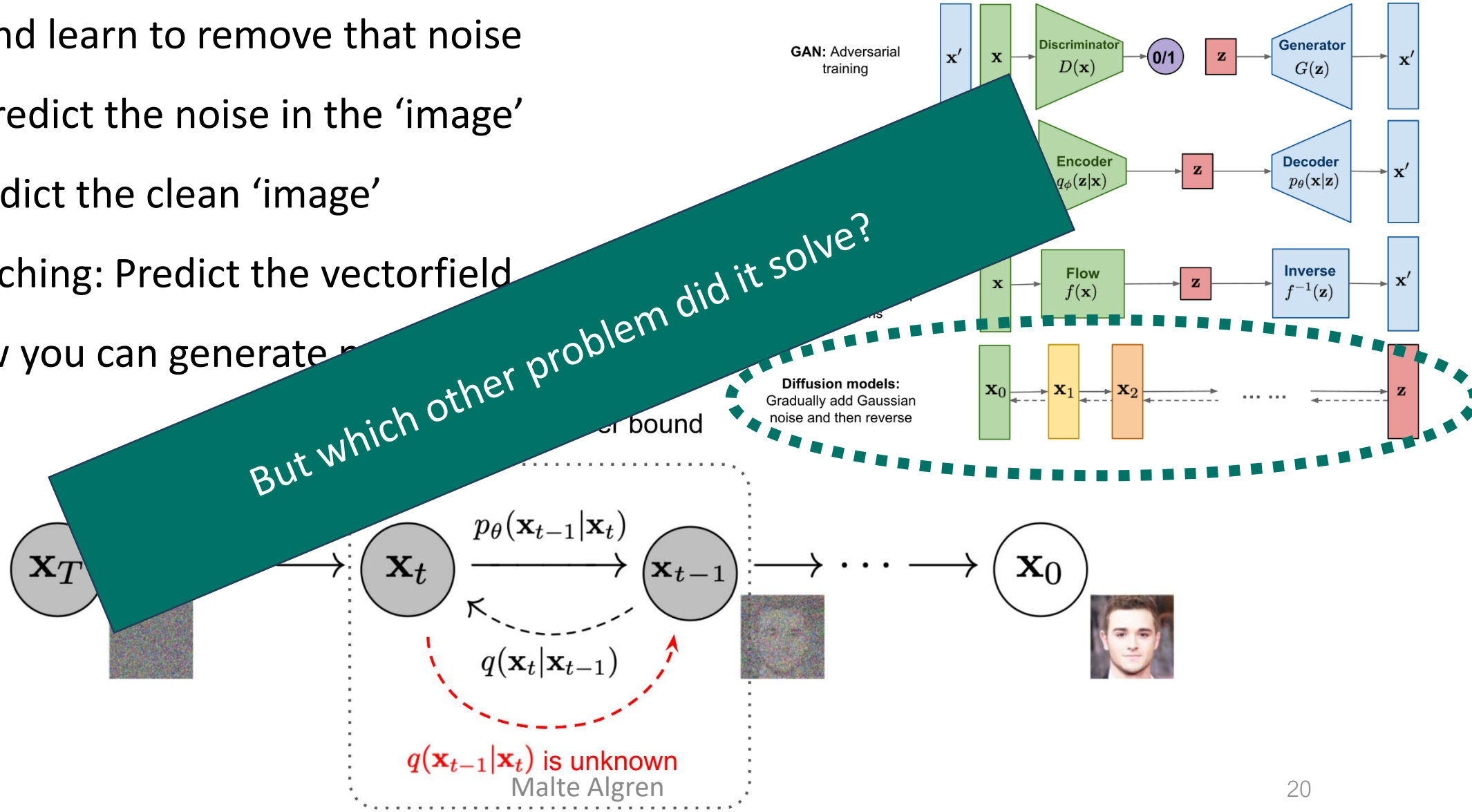


Variational inference: Diffusion

Add noise and learn to remove that noise

- DDPM: Predict the noise in the ‘image’
- EDM: Predict the clean ‘image’
- Flow matching: Predict the vectorfield

And yes now you can generate n



But which other problem did it solve?

Variational inference: Autoencoders

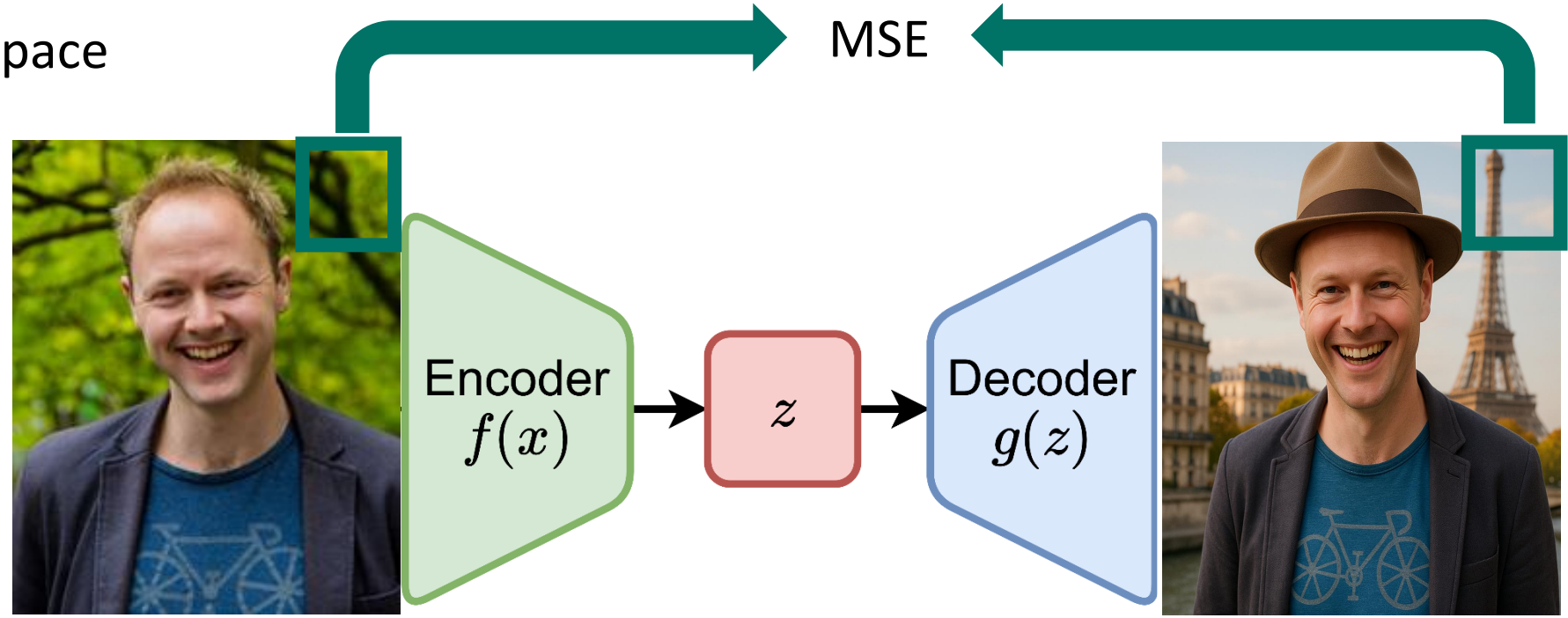
Autoencoders (encoder & decoder)

- Compression in latent space

Advantages:

- Fast (single shot)
- Free latent space

Issues?



Troels C Petersen (2015)
Assistant professor

Troels C Petersen (2004)
PhD student

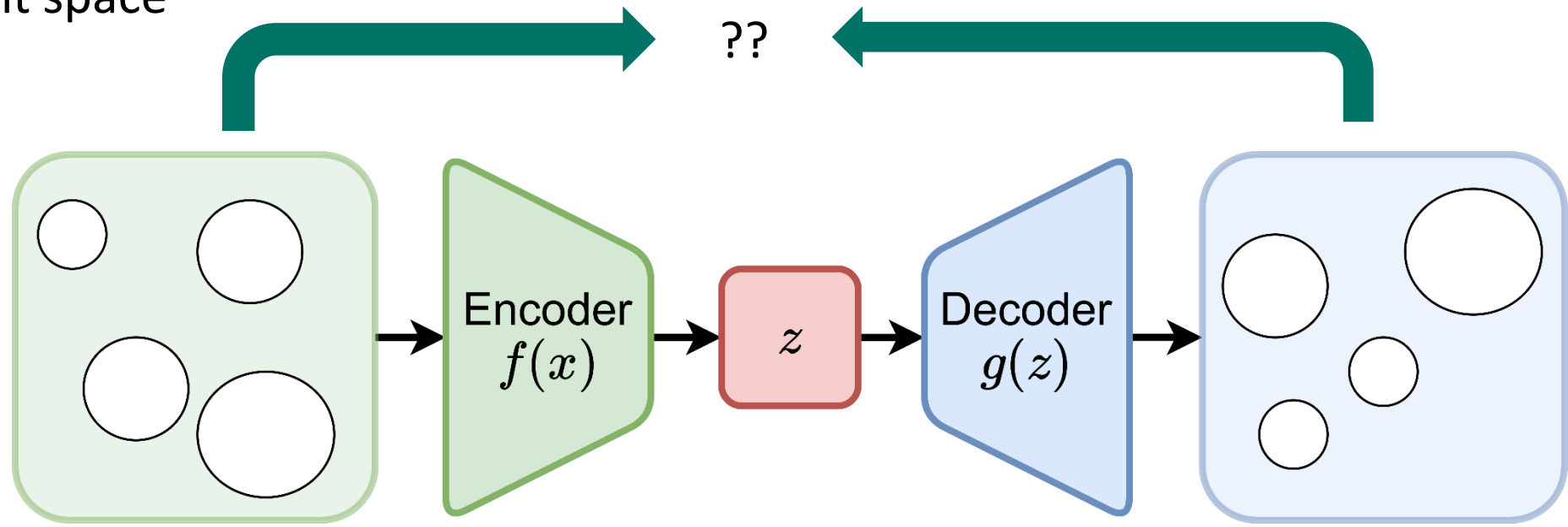
Variational inference: Autoencoders

Autoencoders (encoder & decoder)

- Compression in latent space

Advantages:

- Fast (single shot)
- Free latent space



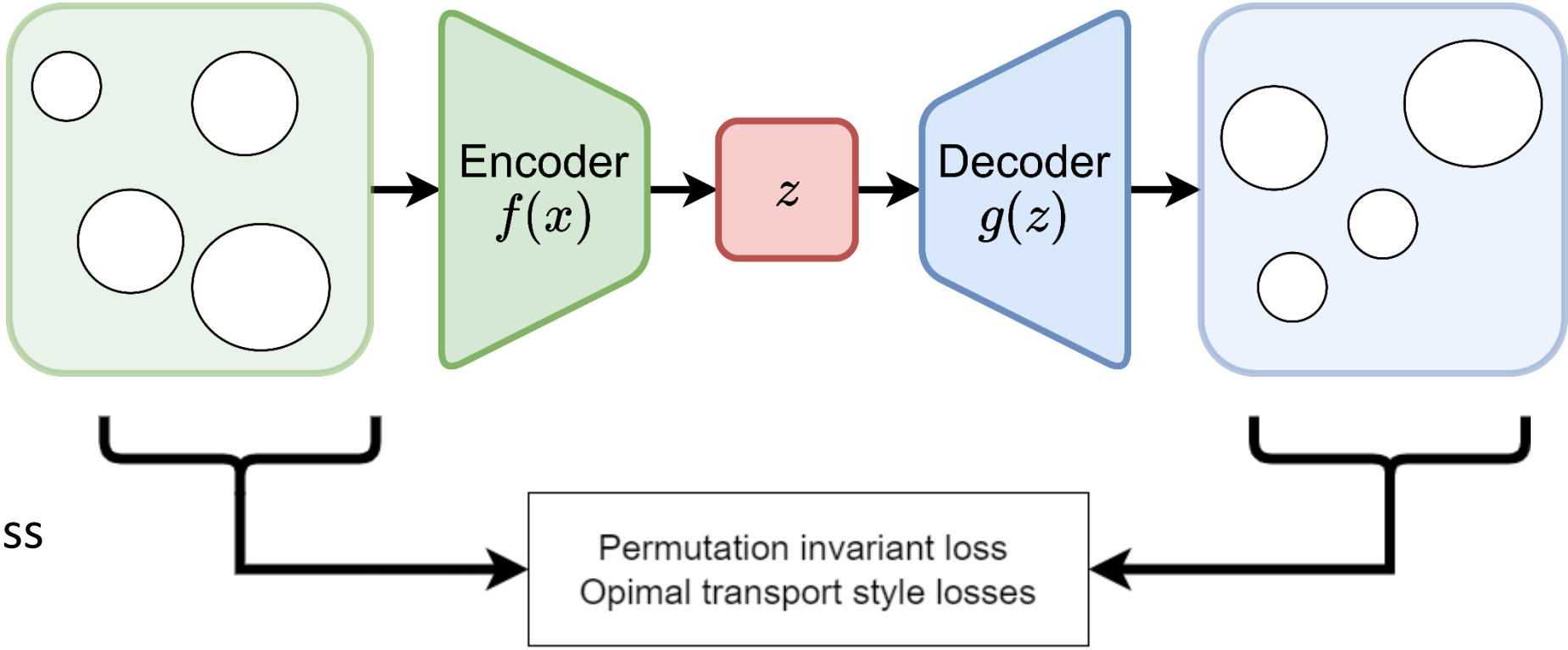
Variational inference: Autoencoders

Autoencoders (encoder & decoder)

- Compression in latent space

Advantages:

- Fast (single shot)
- Free latent space



Distance measure

- Optimal transport loss

Variational inference: Diffusion

Diffusion

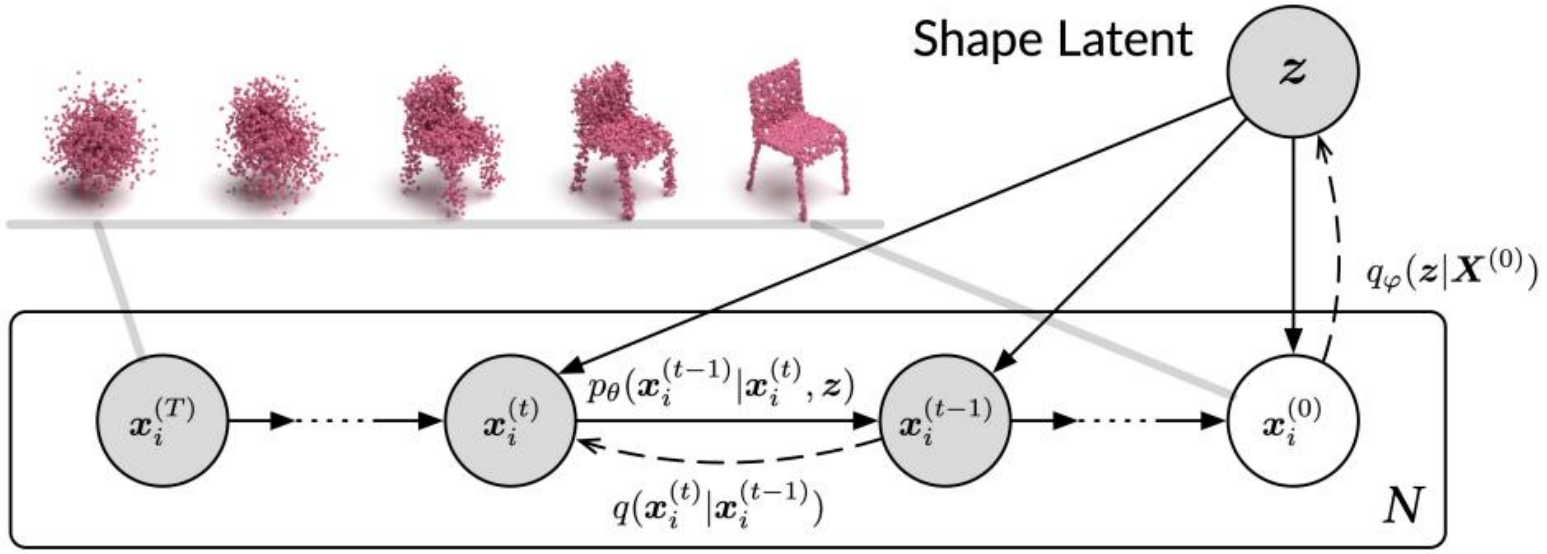
- Adding *small* amount of noise to point cloud
- Because of *small* amount of noise, MSE can still be used
- “Autoregressive flow”

Advantages:

- Permutation invariant
- Works on all structures

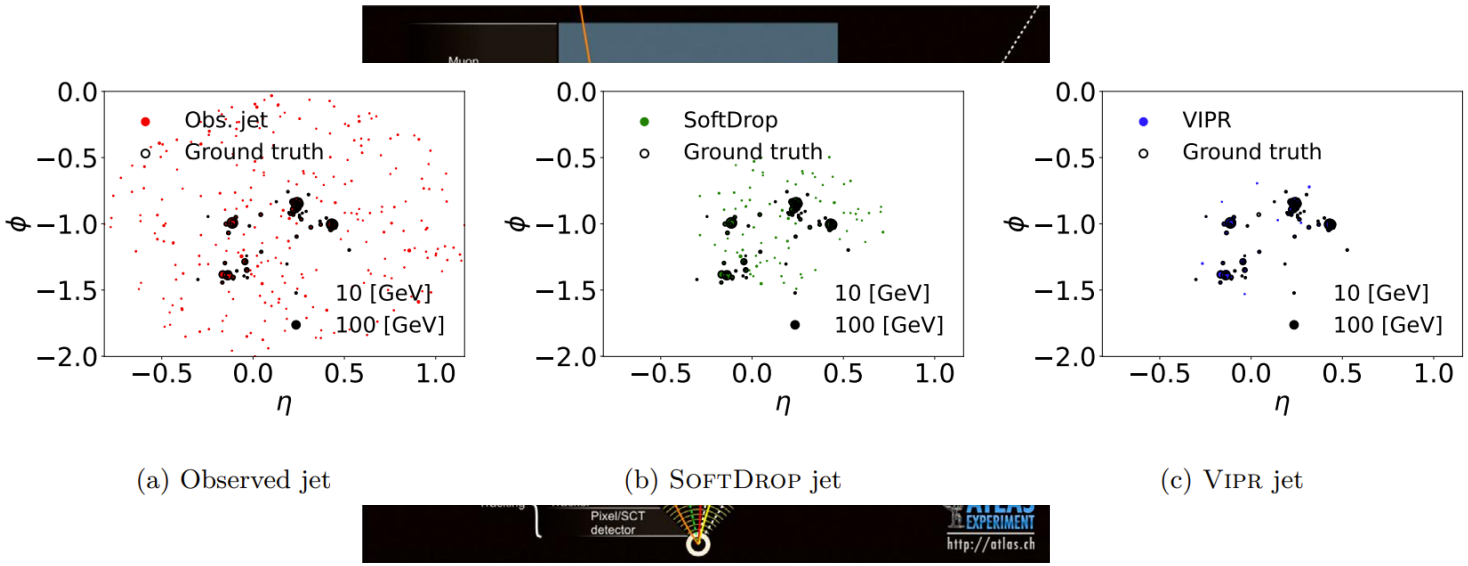
Issues:

- Slow!!!!!



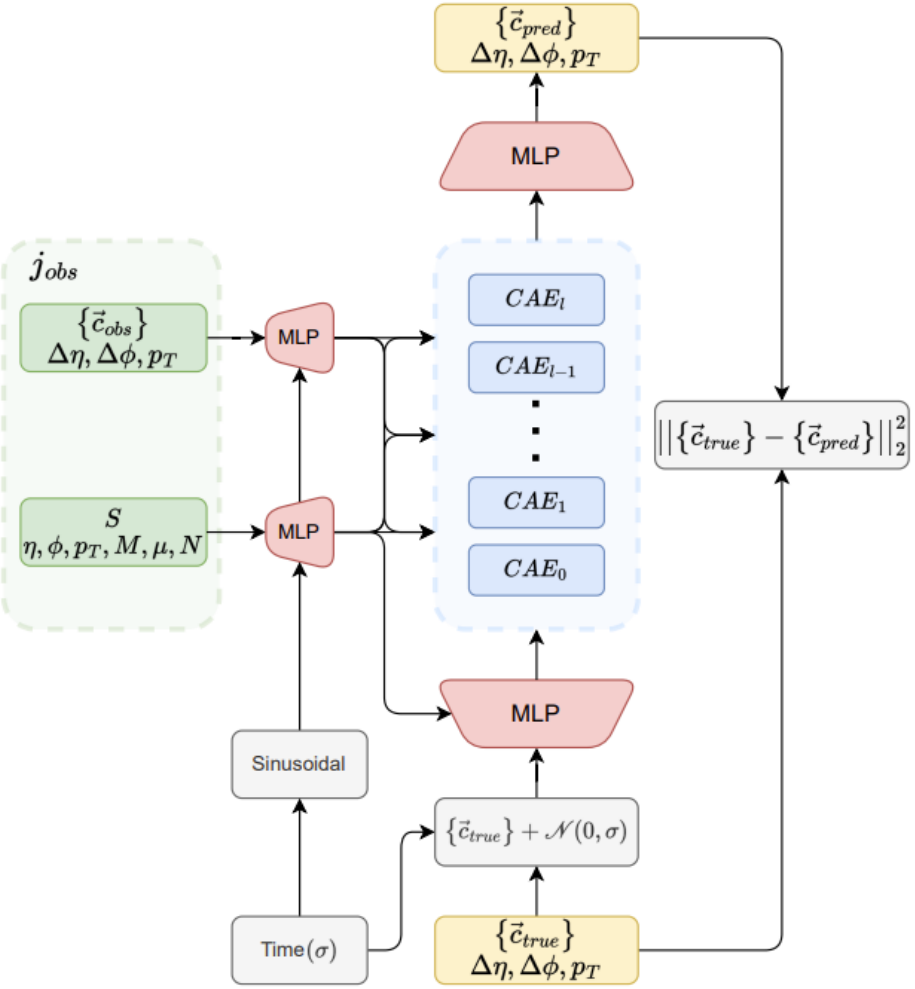
Variational inference: Diffusion

- Opens the door for conditionally generating point cloud
  - Using permutational invariant networks
- For HEP generate PC on constituent level
  - Generate a pileup free jet from an obs. jet



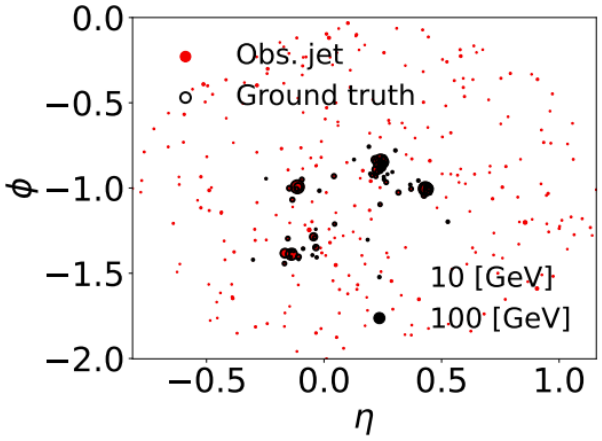
(a) Observed jet (b) SoftDrop jet (c) Vipr jet

EDM training scheme

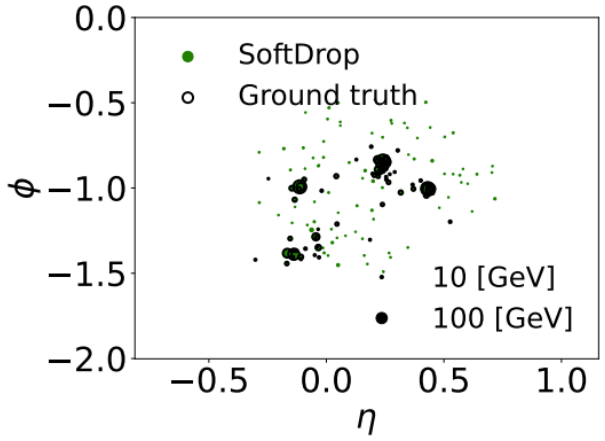


Variational inference: Diffusion

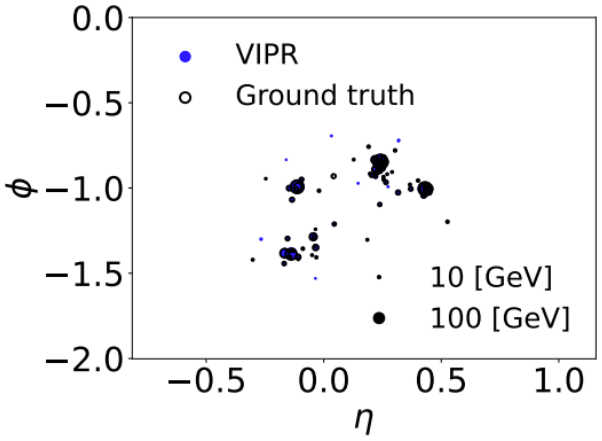
Can be used for pileup (noise) removal at the LHC



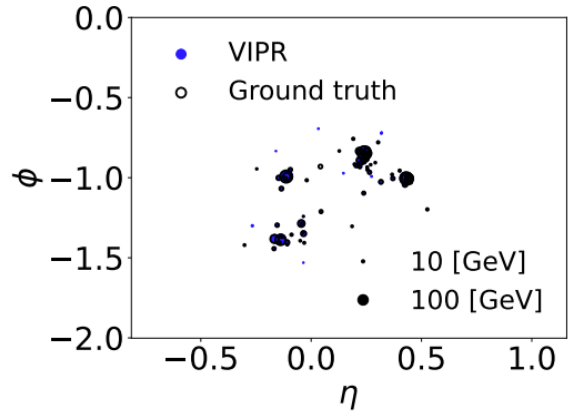
(a) Observed jet

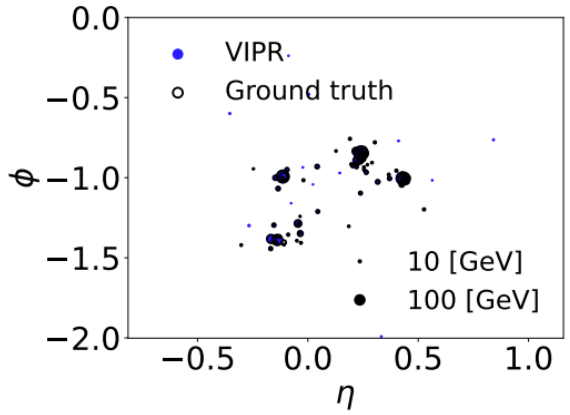


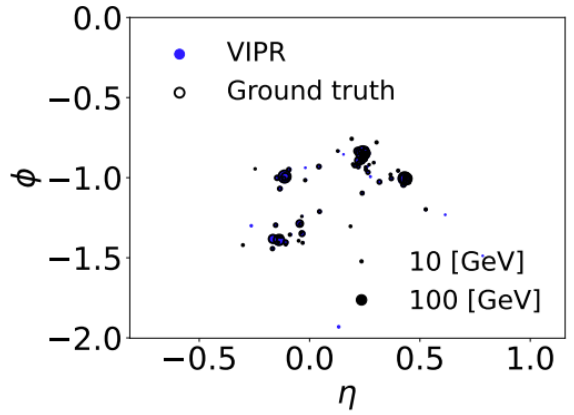
(b) SoftDrop jet



(c) VIPR jet

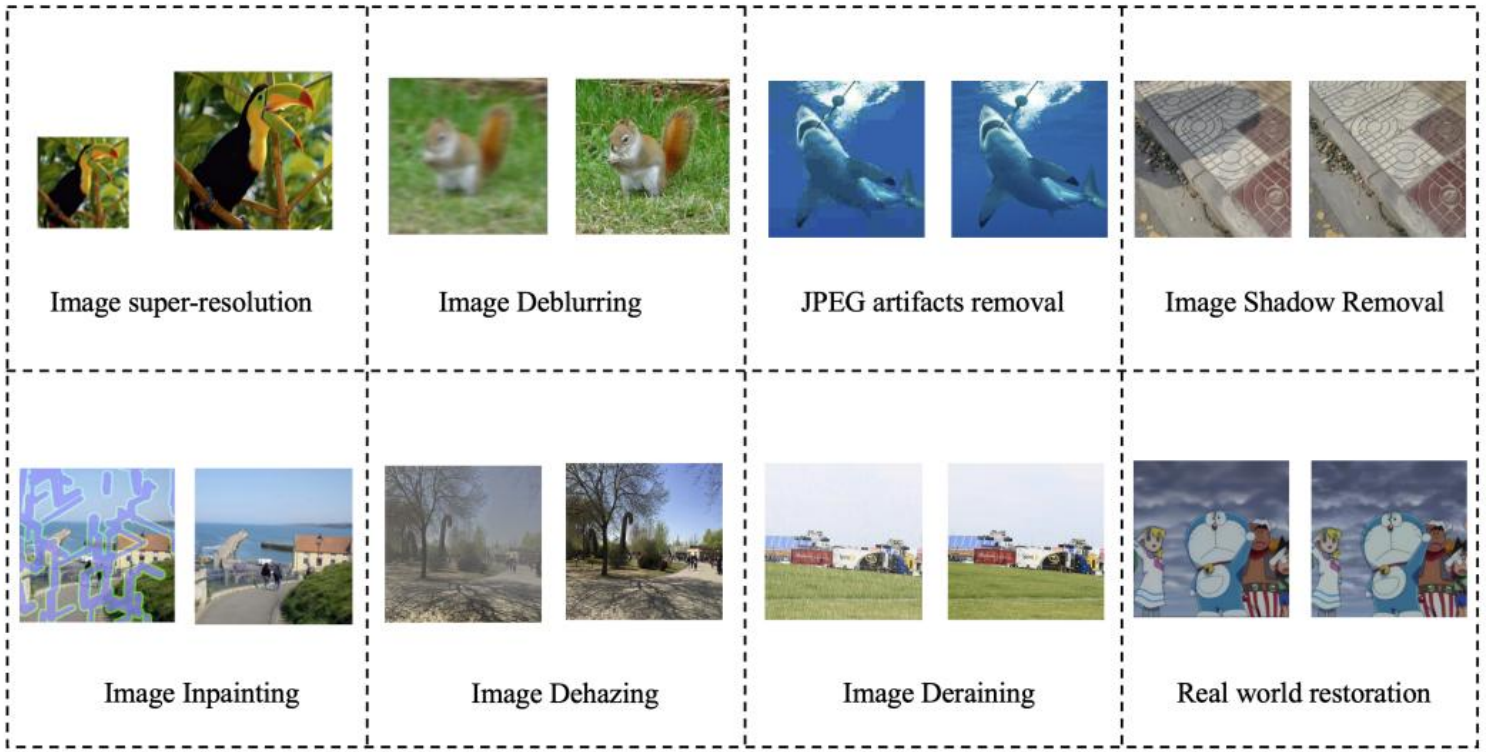


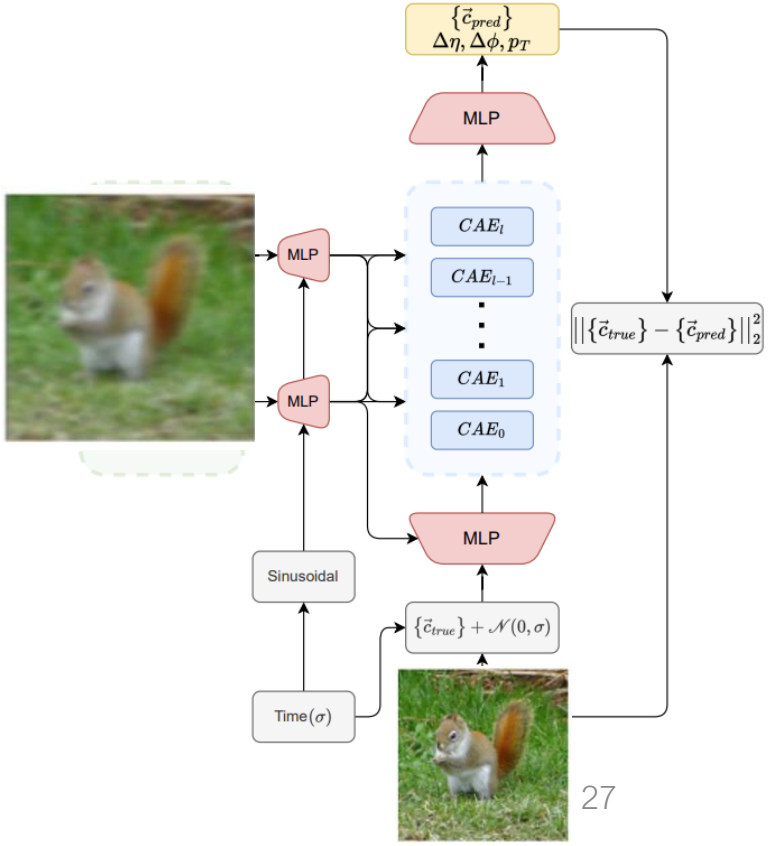




Variational inference: Diffusion

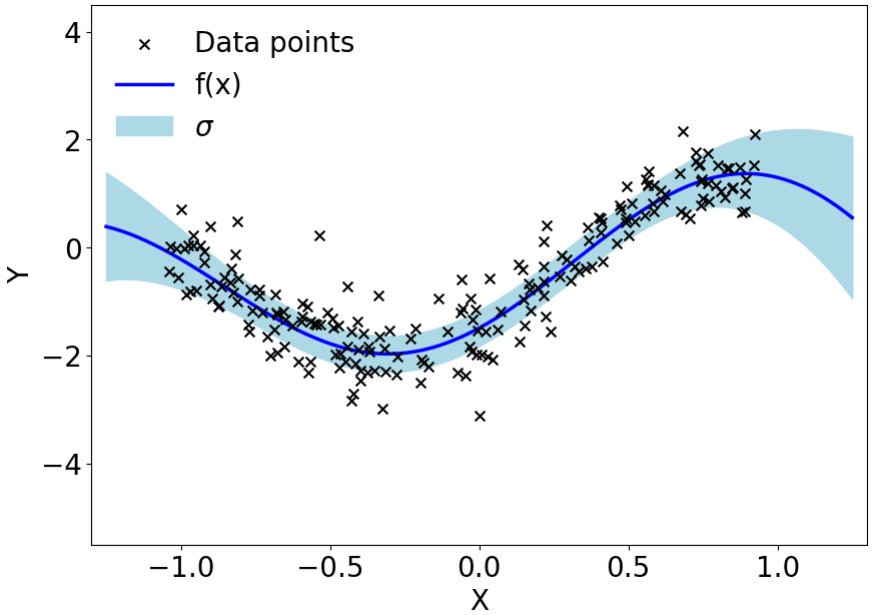
- Used a lot in image enhancement and restoration
  - Condition DiT or Unet on augmented image
  - With simulation, we can do the same (solving inverse problems)

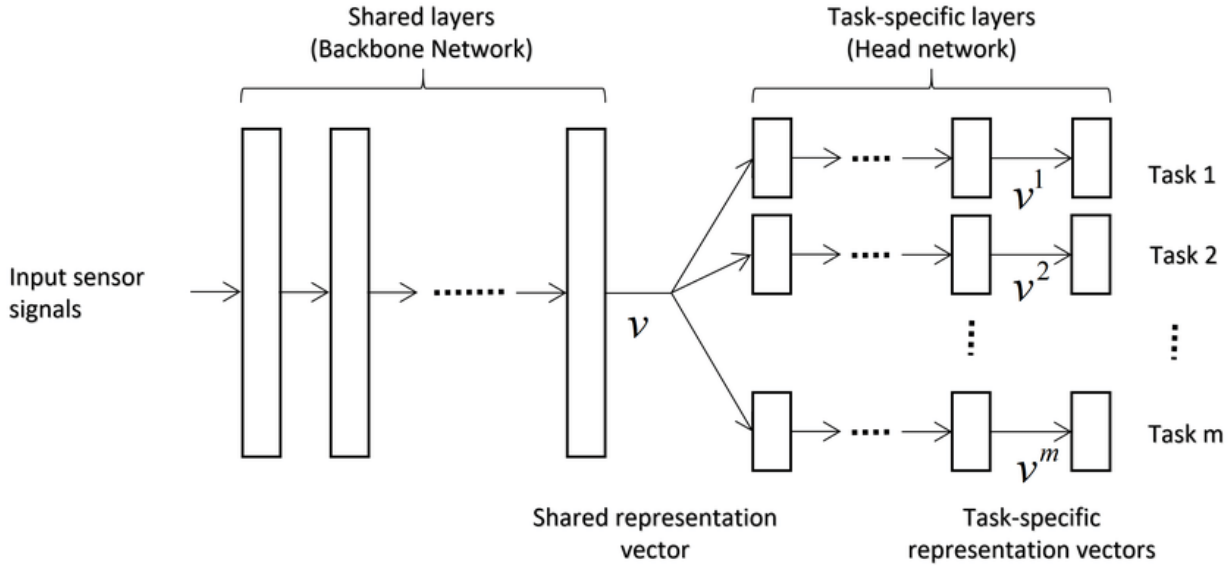




Usefulness of Gaussian likelihood

- Where can regression using gaussian log-likelihood still be useful?
  1. We get an estimate of σ from the network
     - Use σ to estimate where it is difficult to minimize the $(f(x) - y)^2$
- Multi-task learning (MTL): Having σ associate with $\mathcal{L}_i(\cdot)$





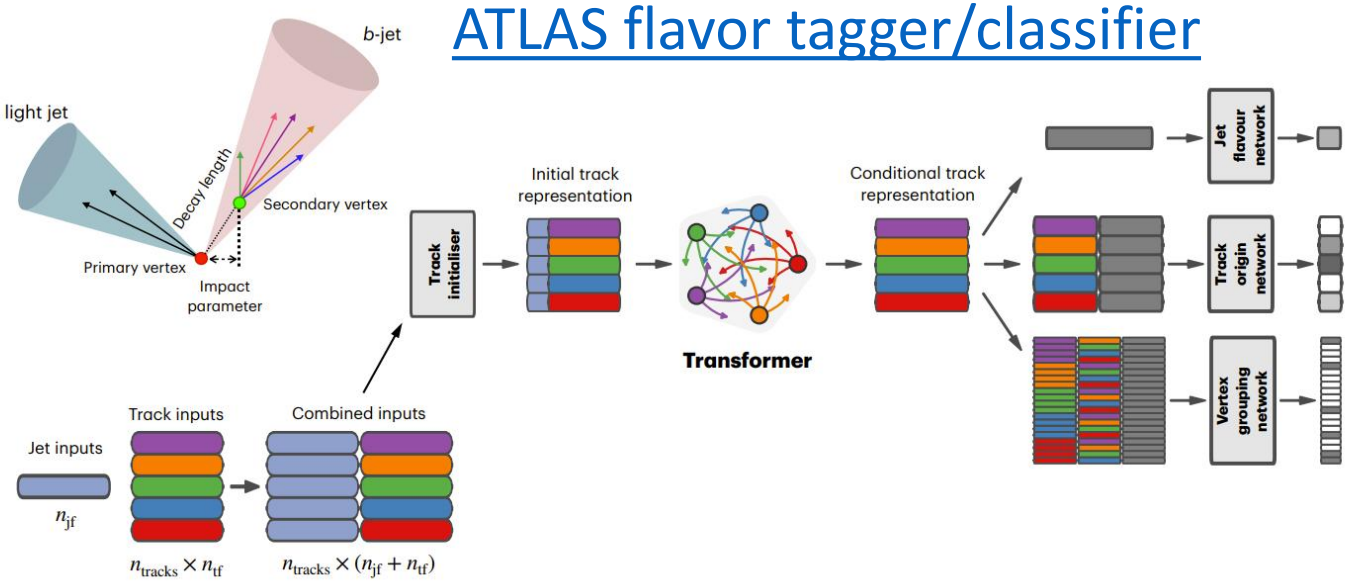
$$\mathcal{L}_{all}(\cdot) = \sum_i \alpha_i \mathcal{L}_i(\cdot)$$

Multi-task learning

- Weighted-static/-dynamic loss: $\mathcal{L}_{all}(\cdot) = \sum_i \alpha_i \mathcal{L}_i(\cdot)$
- Multi-task learning (MTL): Multiple loss functions

1. Regression: $p\left(y_1 | f(x)\right) = \frac{(f(x)-y)^2}{2\sigma^2} + \log(\sigma)$
2. Classification: $p(y_1 | f(x)) = softmax(\frac{1}{\sigma^2} f(x))$
3. Regression + Classification

[ATLAS flavor tagger/classifier]



Multi-task learning

- Weighted-static/-dynamic loss: $\mathcal{L}_{all}(\cdot) = \sum_i \alpha_i \mathcal{L}_i(\cdot)$
- Multi-task learning (MTL): Multiple loss functions

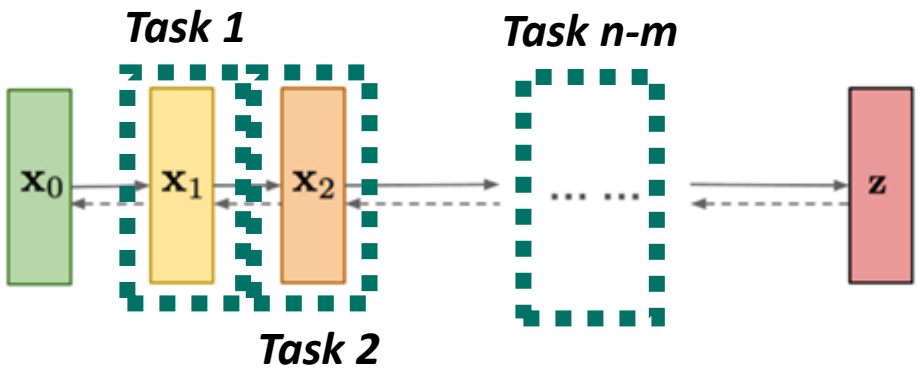
1. Regression: $p\big(y_1|f(x)\big) = \frac{(f(x)-y)^2}{2\sigma^2} + \log(\sigma)$
2. Classification: $p(y_1|f(x)) = softmax(\frac{1}{\sigma^2} f(x))$
3. Regression + Classification

- We have already seen a model that is multi-task learning?
  - Do you know which one?
  - $\log\big(p(y|f(x),\sigma(t)\big) = \frac{(f(x)-y)^2}{2\sigma(t)^2} + \log(\sigma(t))$

Multi-task learning: Diffusion

- Multi-task diffusion model:
  - $\log\left(p(Y|f(X))\right) = \sum_t \frac{(f(x)-Y)^2}{2\sigma(t)^2} + \log(\sigma(t))$
  - Making diffusion invariant:
    - Time sampler
    - Framing/skip connections

Diffusion models:
Gradually add Gaussian noise and then reverse



A lot of researching in framing diffusion model

Table 1: Specific design choices employed by different model families. N is the number of ODE solver iterations that we wish to execute during sampling. The corresponding sequence of time steps is $\{t_0, t_1, \ldots, t_N\}$, where $t_N = 0$. If the model was originally trained for specific choices of N and $\{t_i\}$, the originals are denoted by M and $\{u_j\}$, respectively. The denoiser is defined as $D_\theta(\boldsymbol{x};\sigma) = c_{\text{skip}}(\sigma)\boldsymbol{x} + c_{\text{out}}(\sigma)F_\theta\big(c_{\text{in}}(\sigma)\boldsymbol{x}; c_{\text{noise}}(\sigma)\big)$; F_θ represents the raw neural network layers.

		VP [49]	VE [49]	iDDPM [37] + DDIM [47]	Ours ("EDM")
Sampling (Section 3)					
ODE solver		Euler	Euler	Euler	2nd order Heun
Time steps	$t_{i<N}$	$1 + \frac{i}{N-1}(\epsilon_s - 1)$	$\sigma^2_{\max}\left(\sigma^2_{\min}/\sigma^2_{\max}\right)^{\frac{i}{N-1}}$	$u_{\lfloor j_0 + \frac{M-1-j_0}{N-1} i + \frac{1}{2} \rfloor}$, where $u_M = 0$, $u_{j-1} = \sqrt{\frac{u_j^2+1}{\max(\bar{\alpha}_{j-1}/\bar{\alpha}_j, C_1)} - 1}$	$\left(\sigma_{\max}{}^{\frac{1}{\rho}} + \frac{i}{N-1}\left(\sigma_{\min}{}^{\frac{1}{\rho}} - \sigma_{\max}{}^{\frac{1}{\rho}}\right)\right)^\rho$
Schedule	$\sigma(t)$	$\sqrt{e^{\frac{1}{2}\beta_d t^2 + \beta_{\min} t} - 1}$	$\sqrt{t}$	t	t
Scaling	$s(t)$	$1/\sqrt{e^{\frac{1}{2}\beta_d t^2 + \beta_{\min} t}}$	1	1	1
Network and preconditioning (Section 5)					
Architecture of F_θ		DDPM++	NCSN++	DDPM	(any)
Skip scaling	$c_{\text{skip}}(\sigma)$	1	1	1	$\sigma^2_{\text{data}}/\left(\sigma^2 + \sigma^2_{\text{data}}\right)$
Output scaling	$c_{\text{out}}(\sigma)$	$-\sigma$	σ	$-\sigma$	$\sigma \cdot \sigma_{\text{data}}/\sqrt{\sigma^2_{\text{data}} + \sigma^2}$
Input scaling	$c_{\text{in}}(\sigma)$	$1/\sqrt{\sigma^2 + 1}$	1	$1/\sqrt{\sigma^2 + 1}$	$1/\sqrt{\sigma^2 + \sigma^2_{\text{data}}}$
Noise cond.	$c_{\text{noise}}(\sigma)$	$(M-1)\,\sigma^{-1}(\sigma)$	$\ln(\frac{1}{2}\sigma)$	$M - 1 - \arg\min_j \lvert u_j - \sigma \rvert$	$\frac{1}{4}\ln(\sigma)$
Training (Section 5)					
Noise distribution		$\sigma^{-1}(\sigma) \sim \mathcal{U}(\epsilon_t, 1)$	$\ln(\sigma) \sim \mathcal{U}(\ln(\sigma_{\min}), \ln(\sigma_{\max}))$	$\sigma = u_j,\ \ j \sim \mathcal{U}\{0, M-1\}$	$\ln(\sigma) \sim \mathcal{N}(P_{\text{mean}}, P^2_{\text{std}})$
Loss weighting	$\lambda(\sigma)$	$1/\sigma^2$	$1/\sigma^2$	$1/\sigma^2$ (note: *)	$\left(\sigma^2 + \sigma^2_{\text{data}}\right)/(\sigma \cdot \sigma_{\text{data}})^2$
Parameters		$\beta_d = 19.9, \beta_{\min} = 0.1$ $\epsilon_s = 10^{-3}, \epsilon_t = 10^{-5}$ $M = 1000$	$\sigma_{\min} = 0.02$ $\sigma_{\max} = 100$	$\bar{\alpha}_j = \sin^2(\frac{\pi}{2}\frac{j}{M(C_2+1)})$ $C_1 = 0.001, C_2 = 0.008$ $M = 1000, j_0 = 8^\dagger$	$\sigma_{\min} = 0.002, \sigma_{\max} = 80$ $\sigma_{\text{data}} = 0.5, \rho = 7$ $P_{\text{mean}} = -1.2, P_{\text{std}} = 1.2$

* iDDPM also employs a second loss term L_{vlb} † In our tests, $j_0 = 8$ yielded better FID than $j_0 = 0$ used by iDDPM

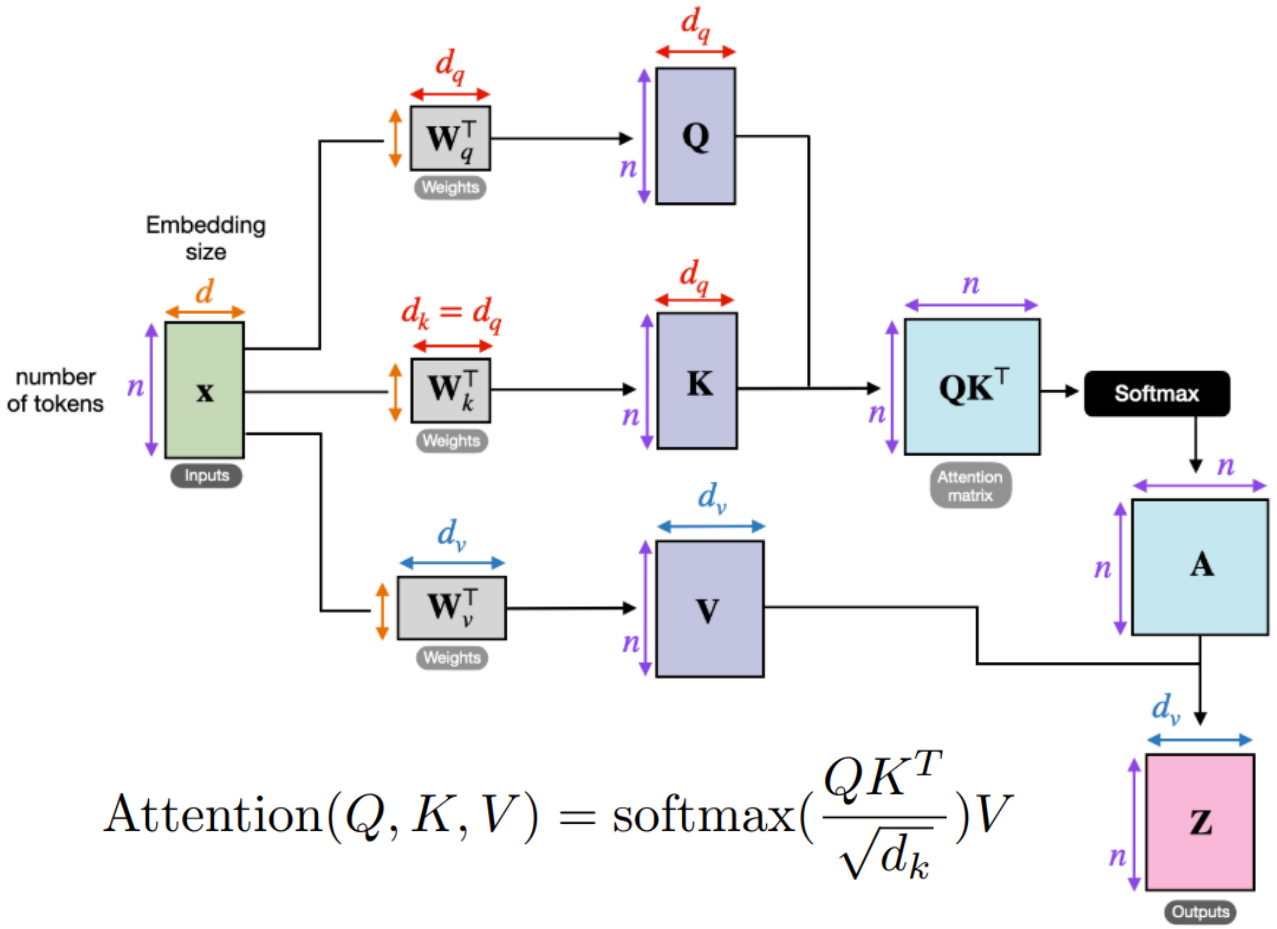
Any questions?

Otherwise, we go to architectures and

ideas on how to handle big ass point-clouds

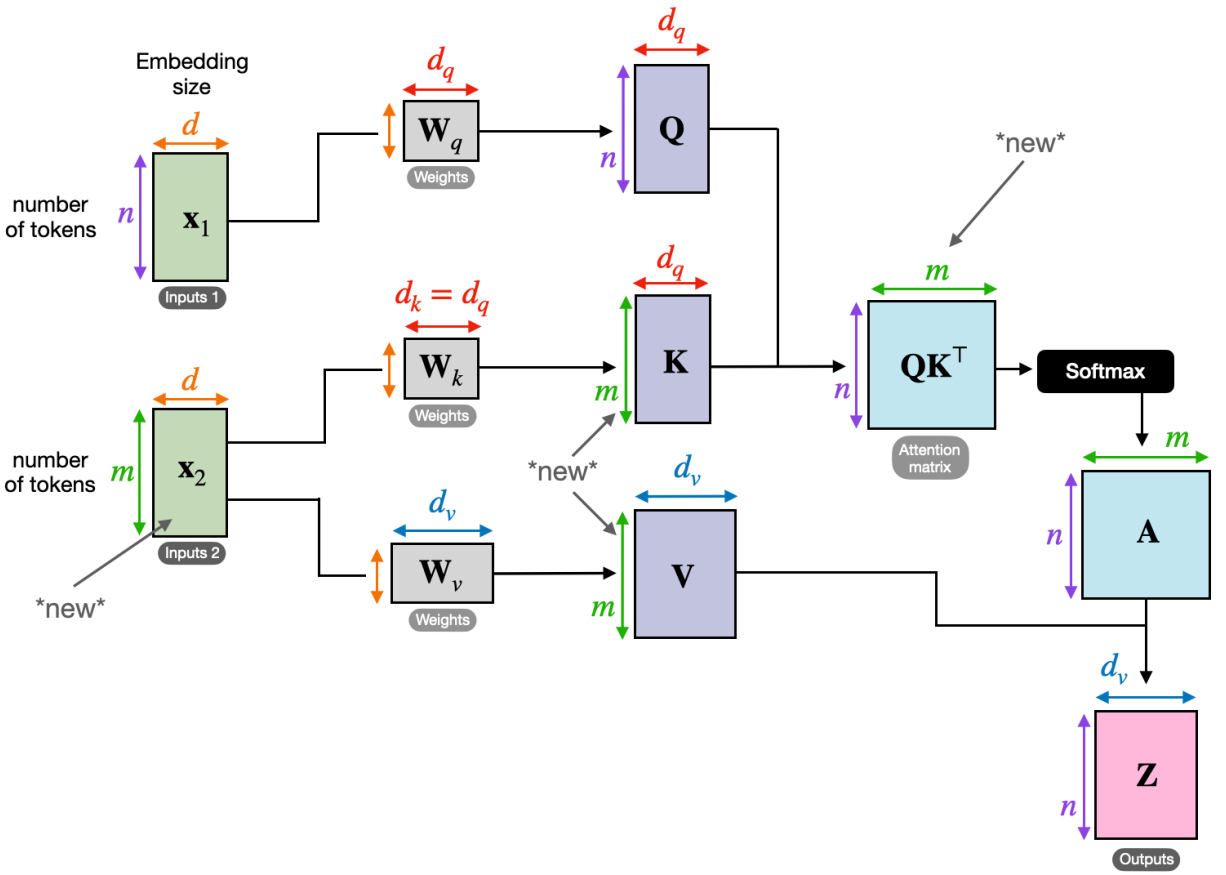
Transformers

Self attention (SA): n**2



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Cross attention (CA): n*m



Transformers

Attention is all you need

- Encoder with self attention (SA)
- Decoder with SA + cross attention (CA)

Pre-LN ===>

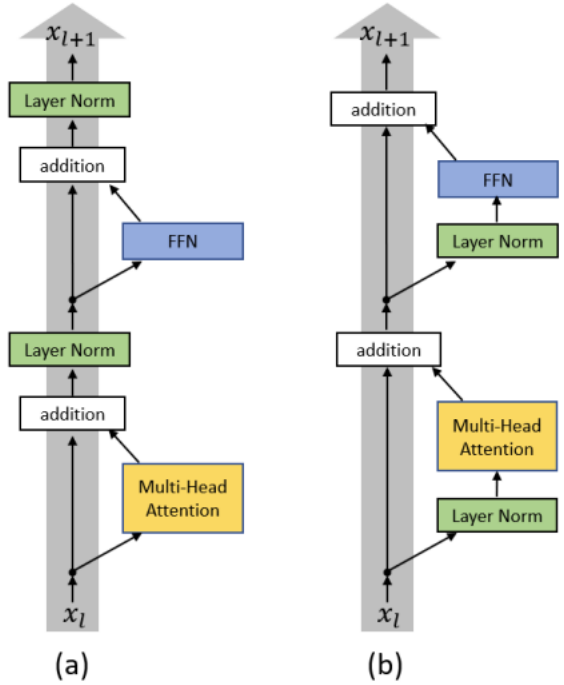
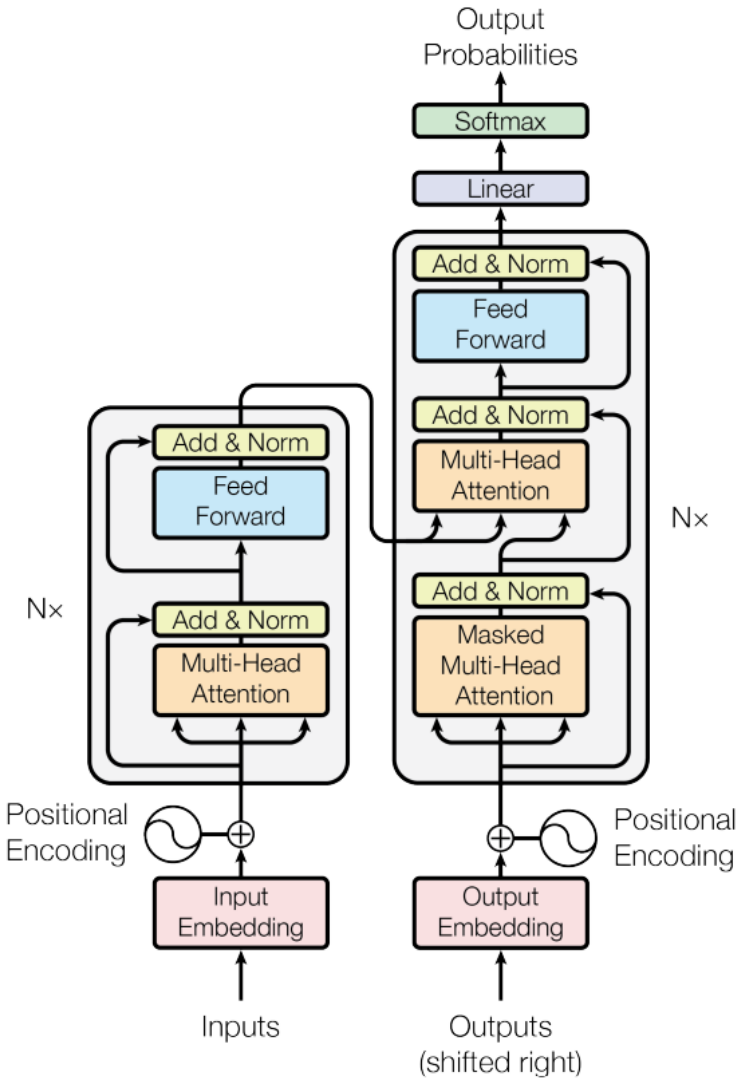
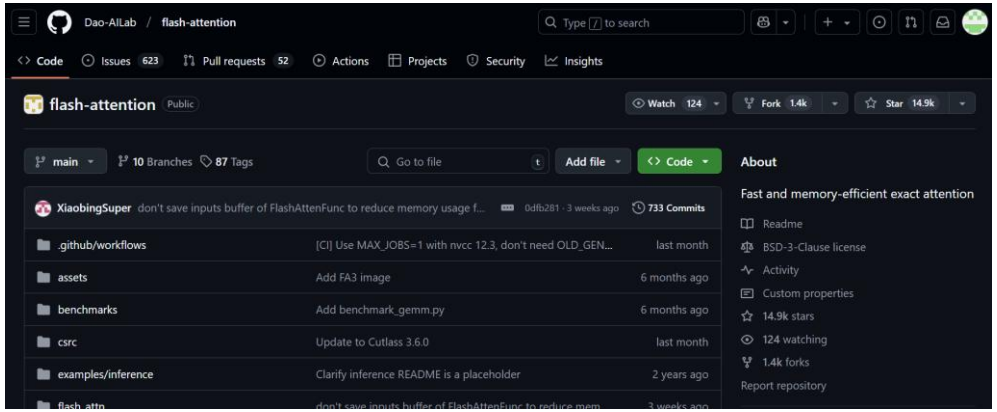


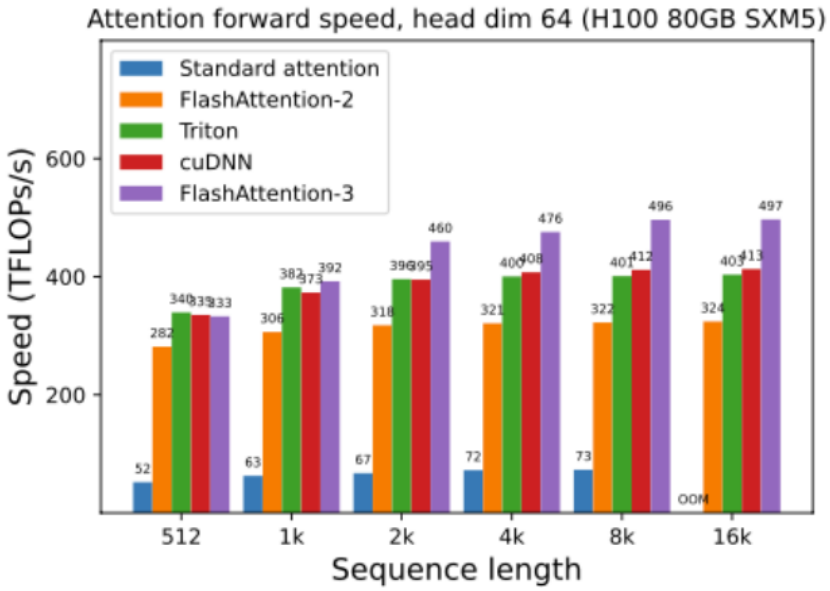
Figure 1. (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

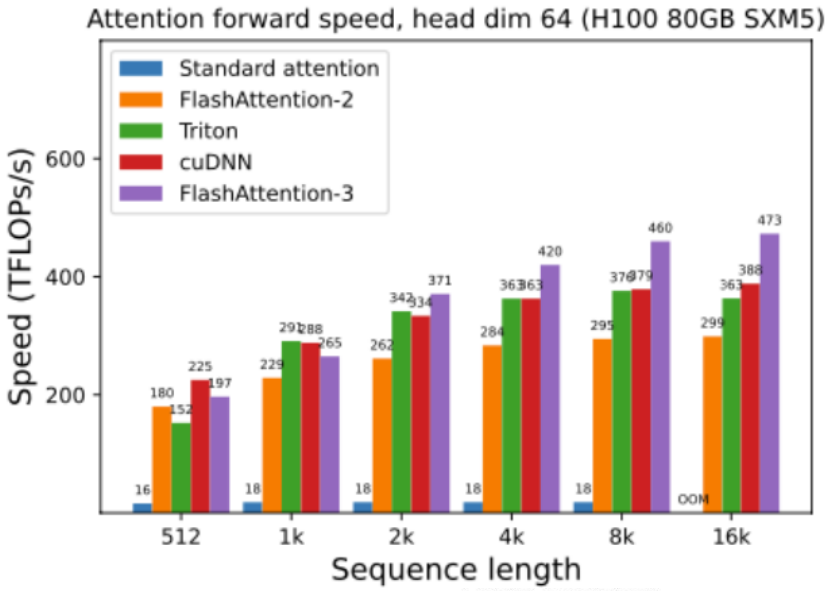


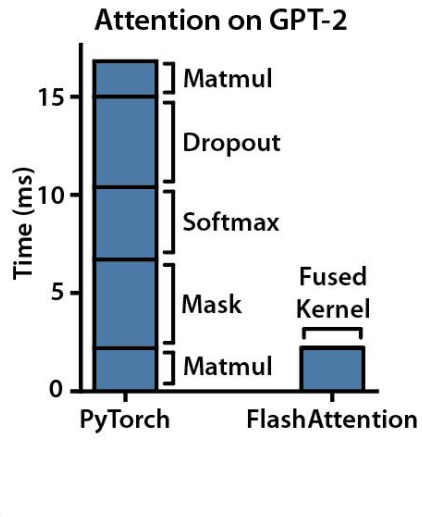
Transformers: FlashAttn

- Speeding up scaled dot product attention
  - Hardware-aware attention (fancy cuda)
- Benchmarked on NLP tasks
- ***Require ampere GPUs***





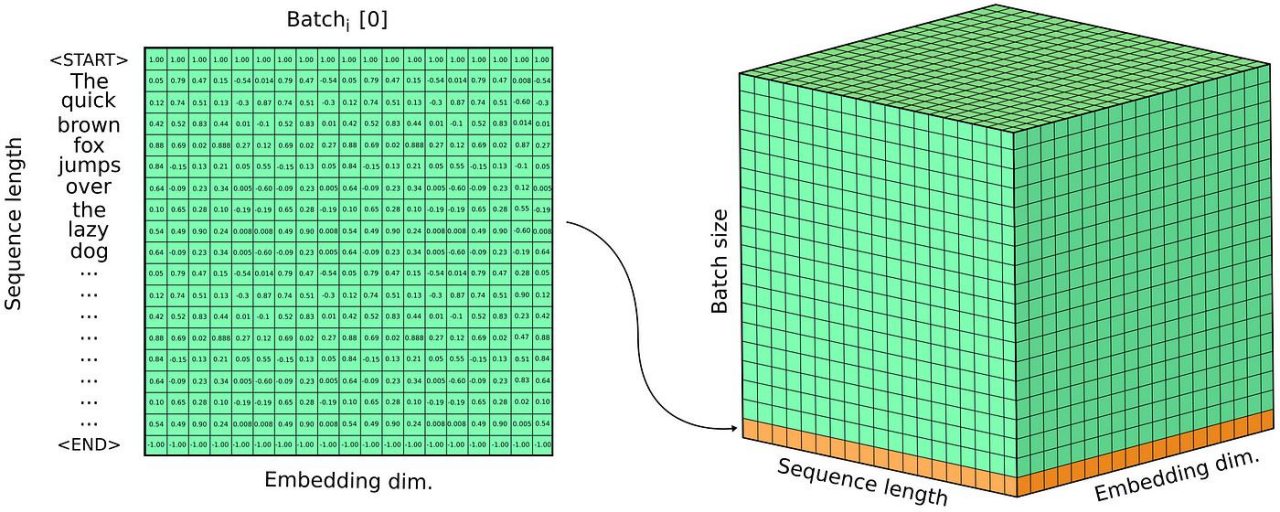




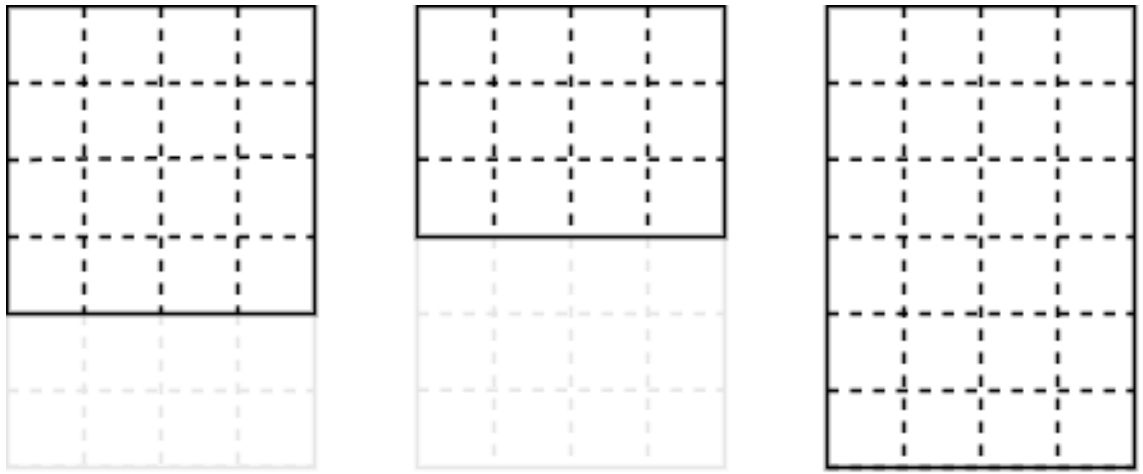
Transformers: FlashAttn

- Speeding up scaled dot product attention
  - Hardware-aware attention (fancy cuda)
- Benchmarked on NLP tasks
- ***Require ampere GPUs***

PyTorch batches

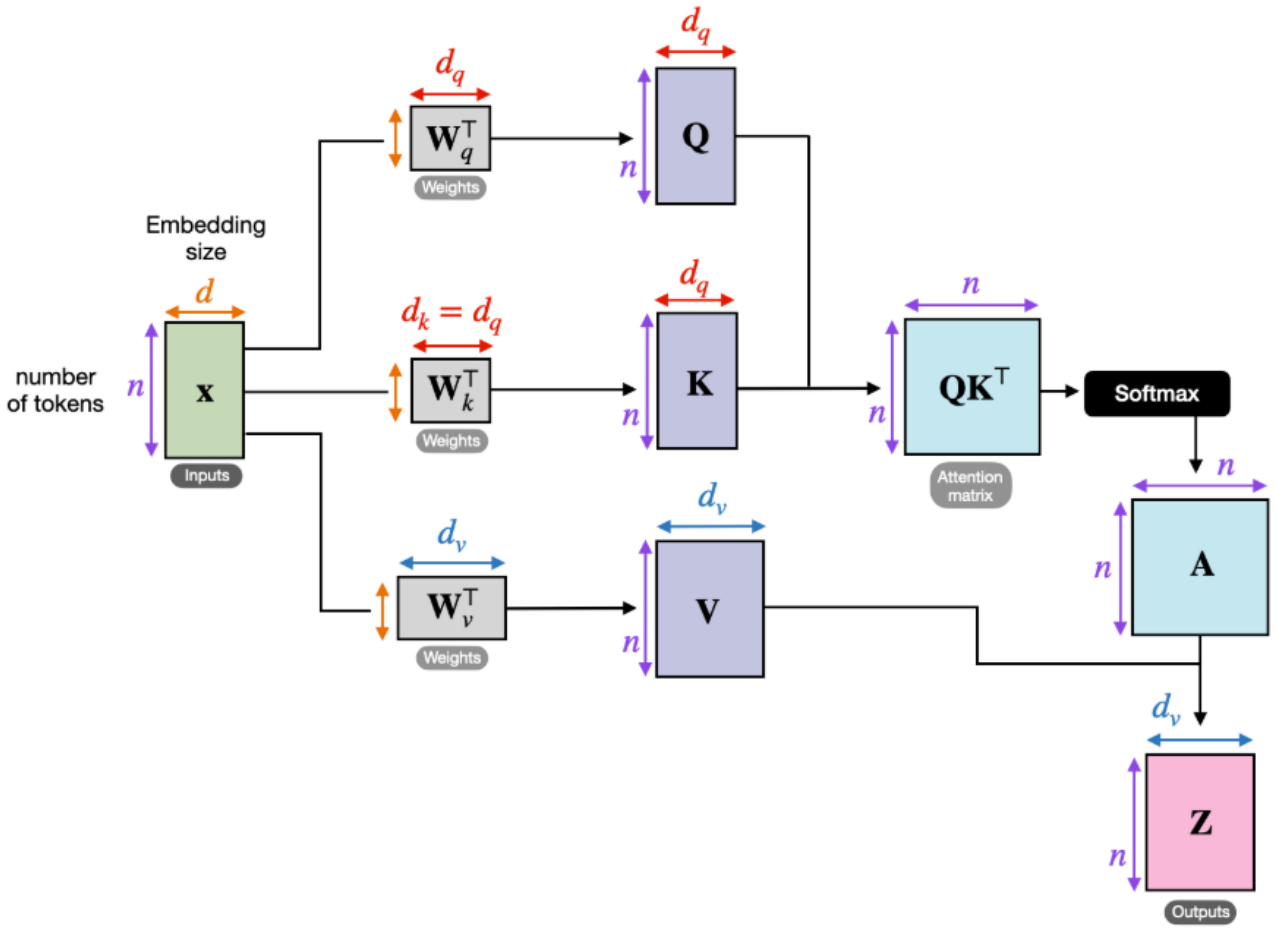


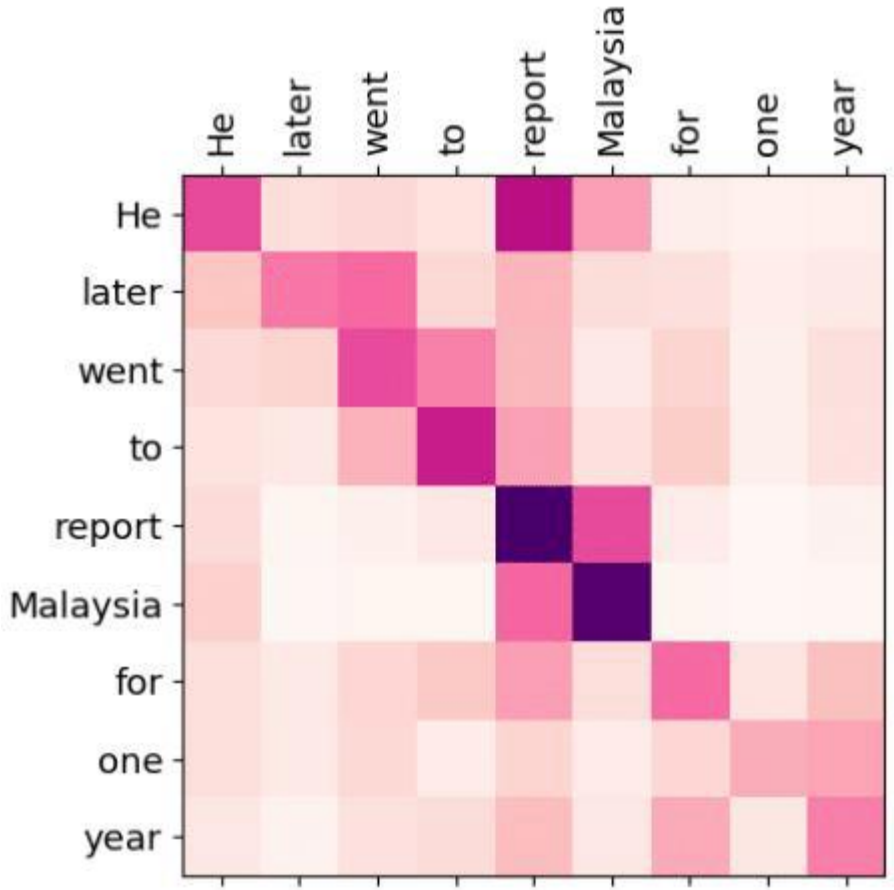
FlashAttn batches



Transformers: Mixture-of-Depths

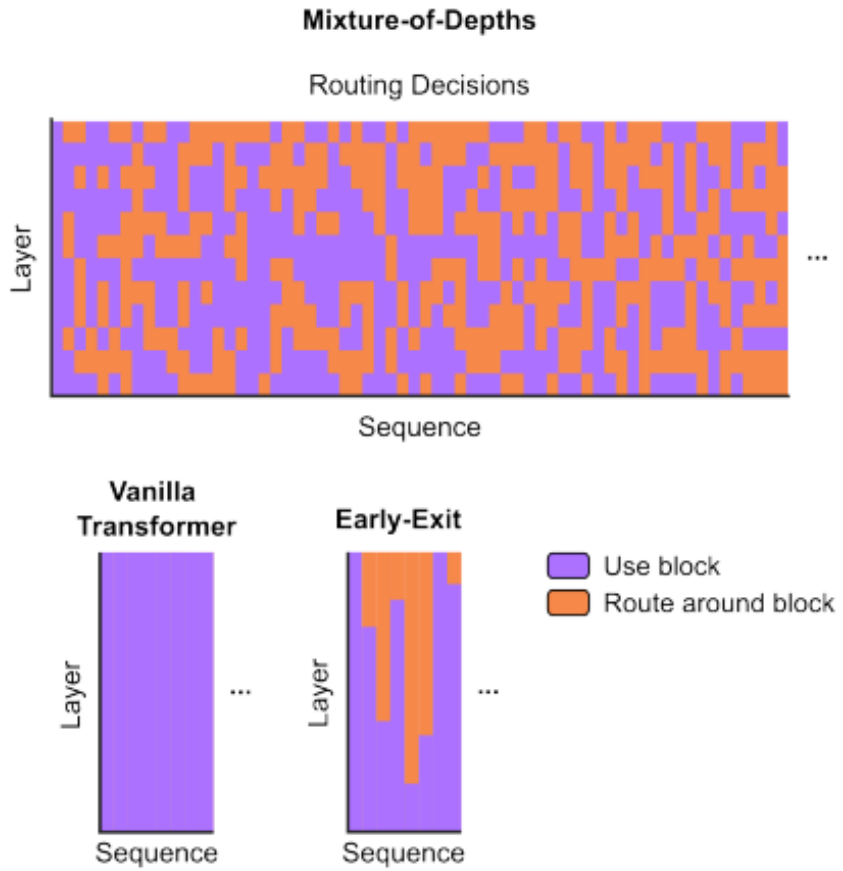
Mixture-of-Depths: Dynamically allocating compute in transformer-based language models:





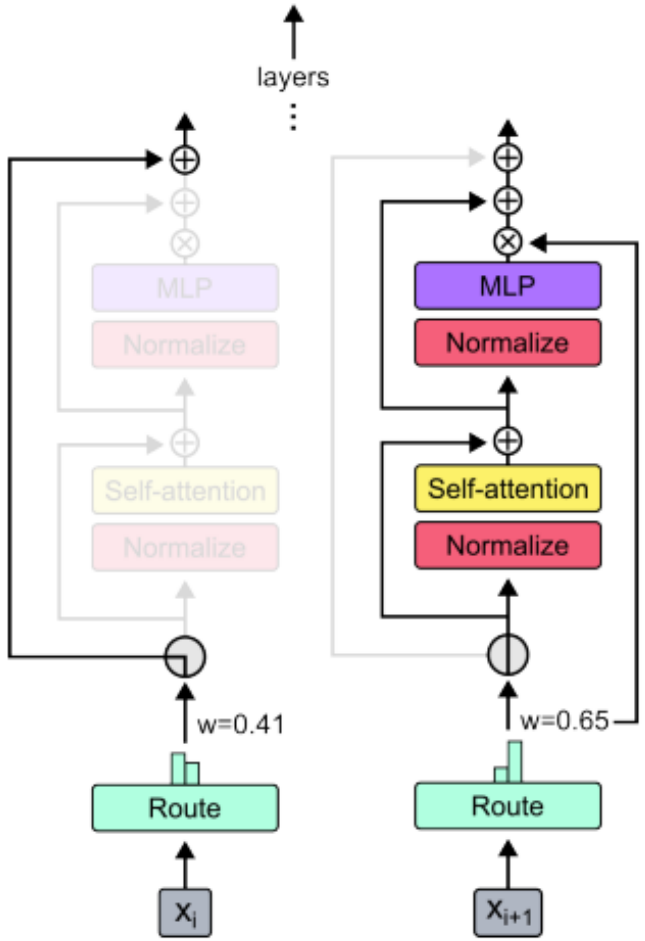
Transformers: Mixture-of-Depths

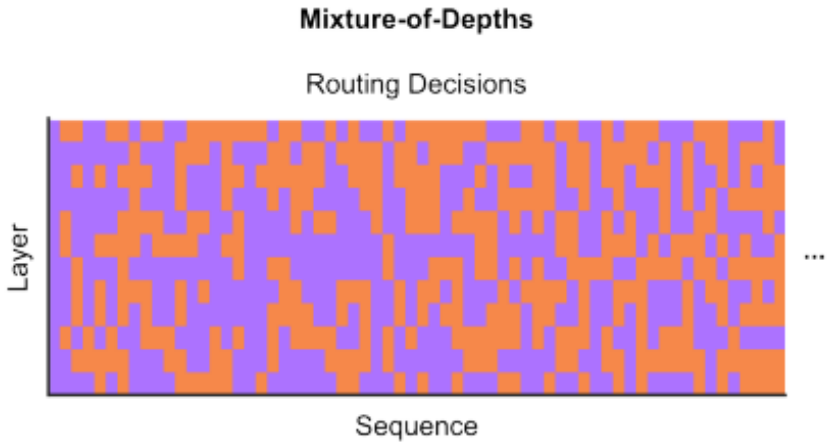
Mixture-of-Depths: Dynamically allocating compute in transformer-based language models:

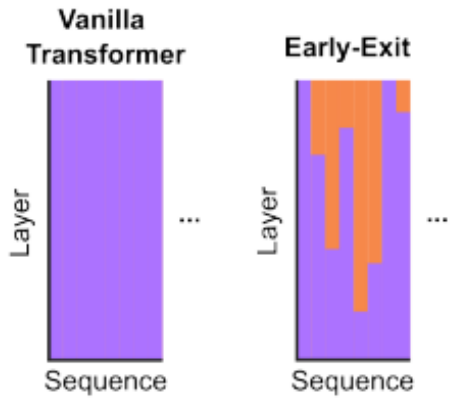


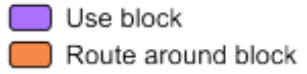
Transformers: Mixture-of-Depths

Mixture-of-Depths: Dynamically allocating compute in transformer-based language models:





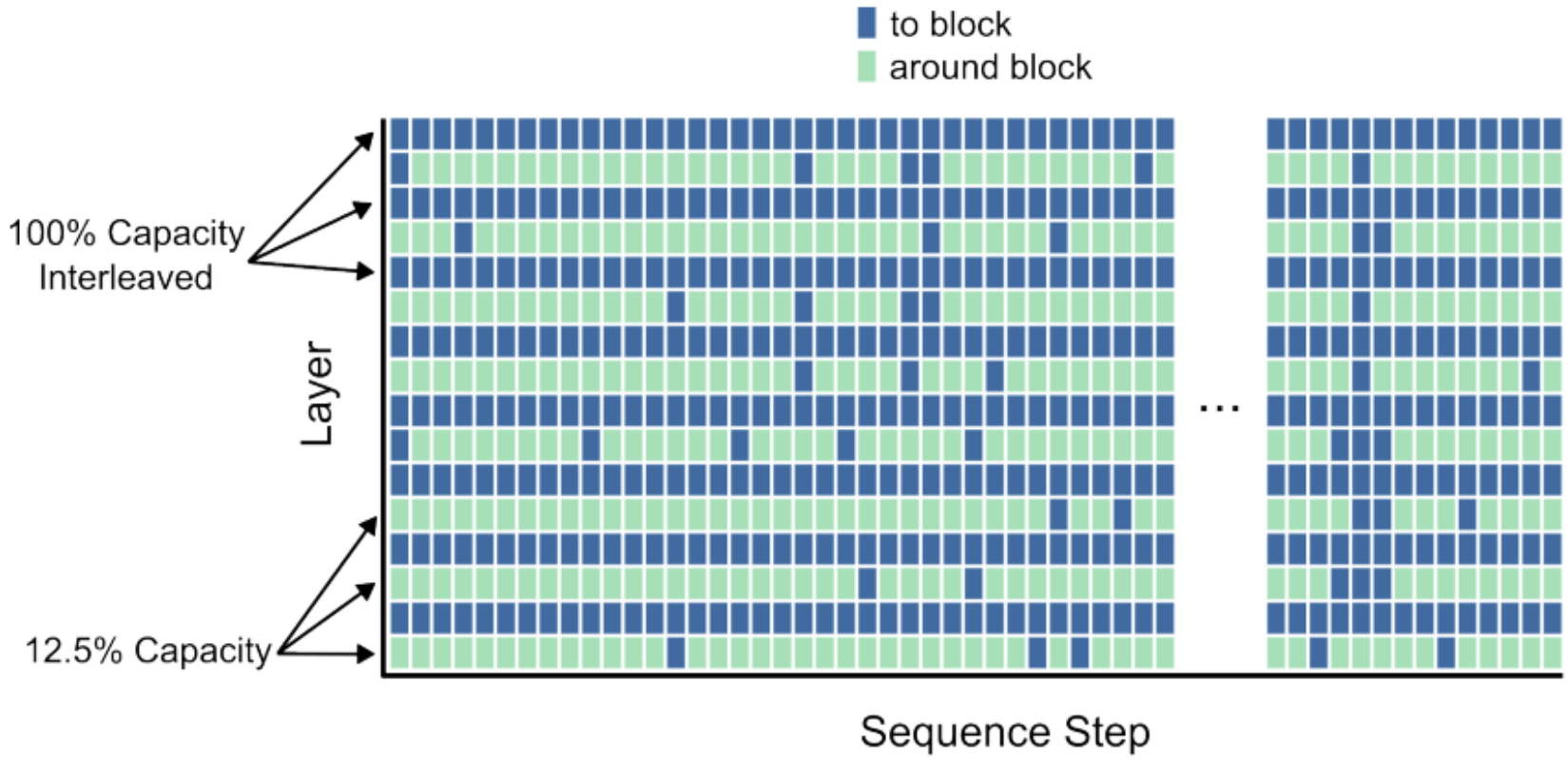


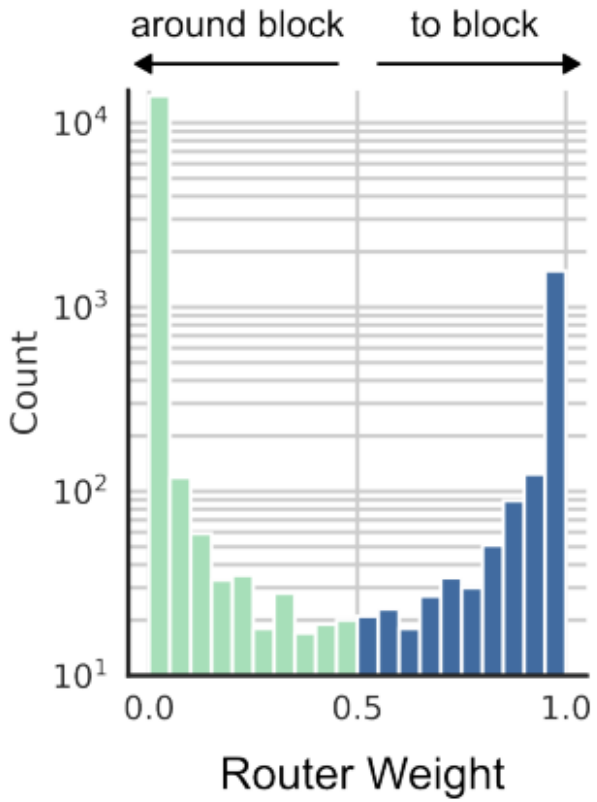


$$x_i^{l+1} = \begin{cases} r_i^l f_i(\tilde{X}^l) + x_i^l, & \text{if } r_i^l > P_\beta(R^l) \\ x_i^l, & \text{if } r_i^l < P_\beta(R^l) \end{cases}$$

Transformers: Mixture-of-Depths

Mixture-of-Depths: Dynamically allocating compute in transformer-based language models:

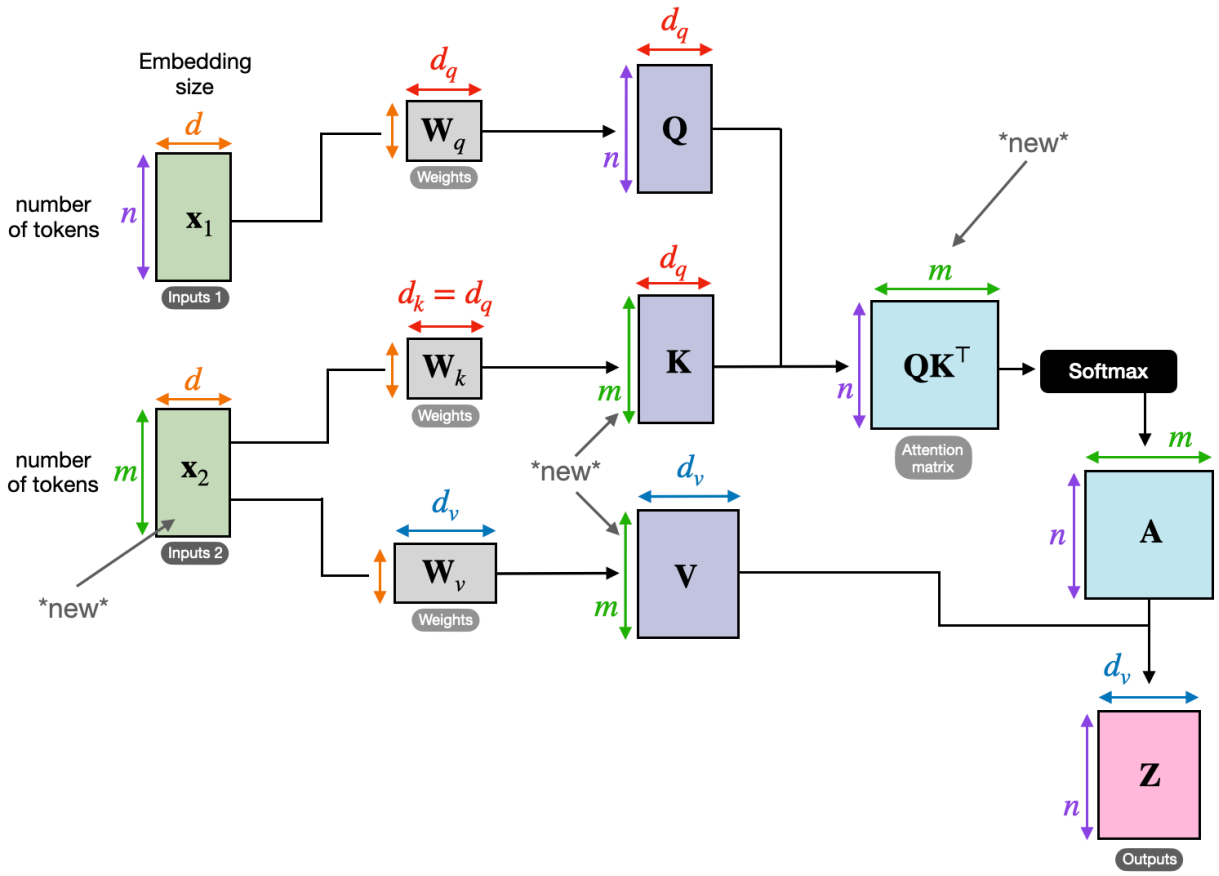




Transformers: PerceiverIO

Lets said we have a pc of ~2000 points and want to exchange information between points:

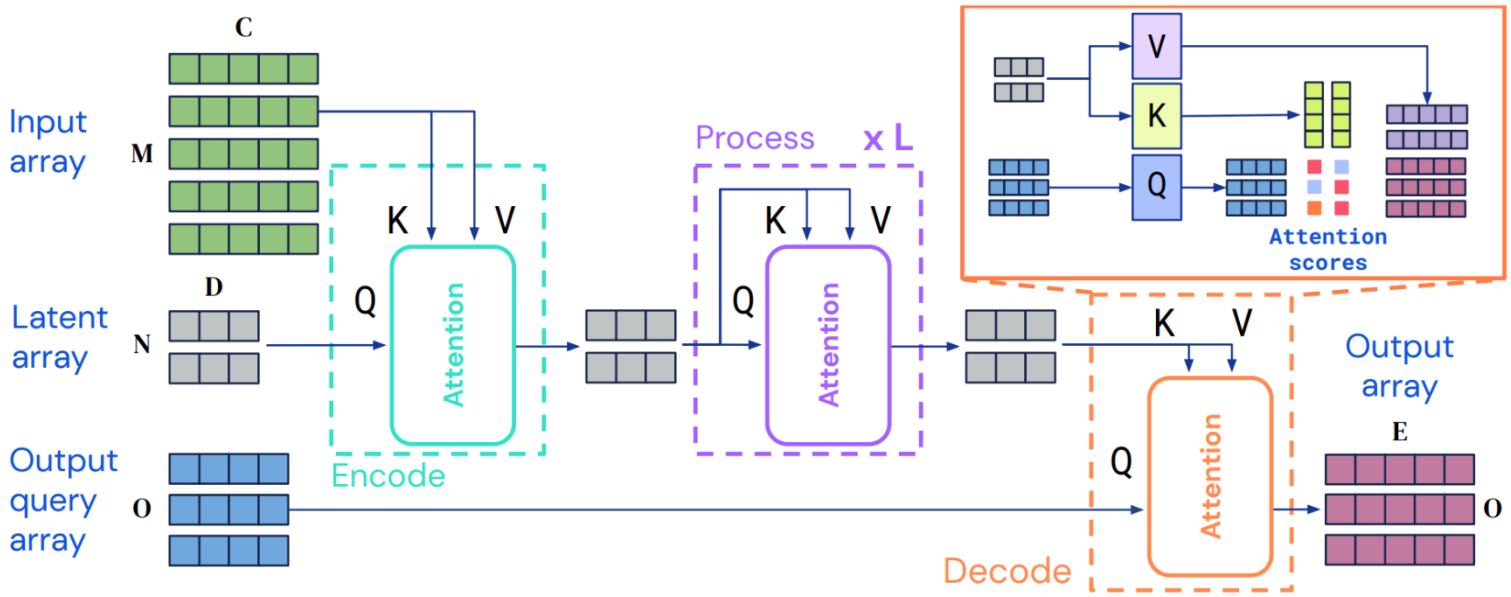
- SA is expensive (2000**2)
- CA would be (2000*m)
  - But what can m be?



Transformers: PerceiverIO

Lets said we have a pc of ~2000 points and want to exchange information between points:

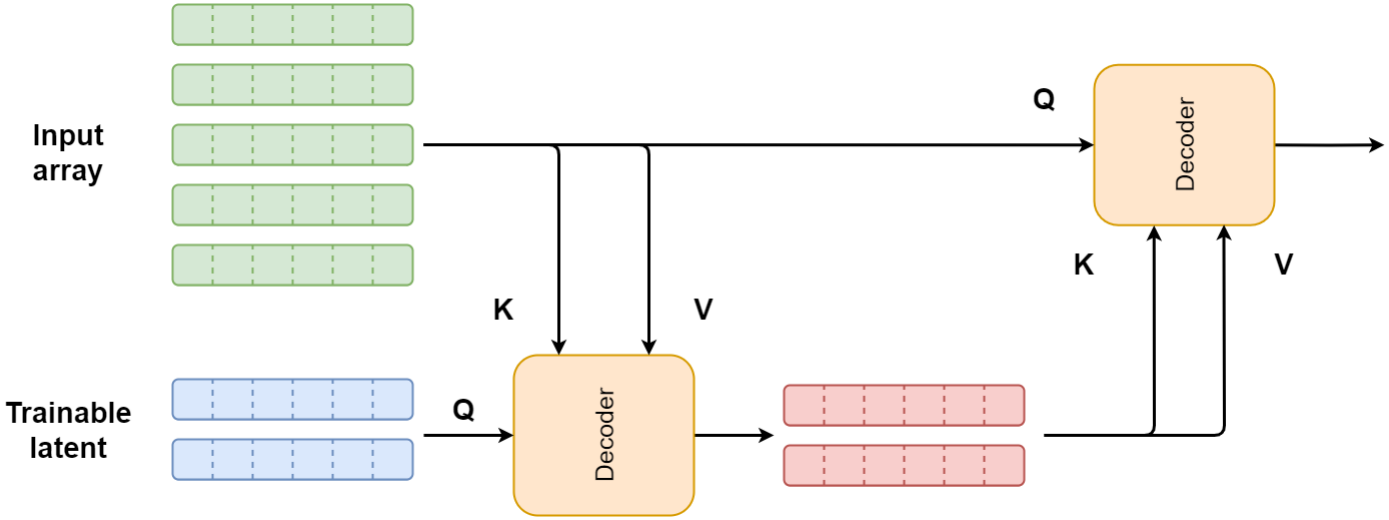
- SA is expensive (2000**2)
- CA would be (2000*m)
  - But what can m be?
- Trainable point cloud
  - w/ active gradient



Transformers: PerceiverIO

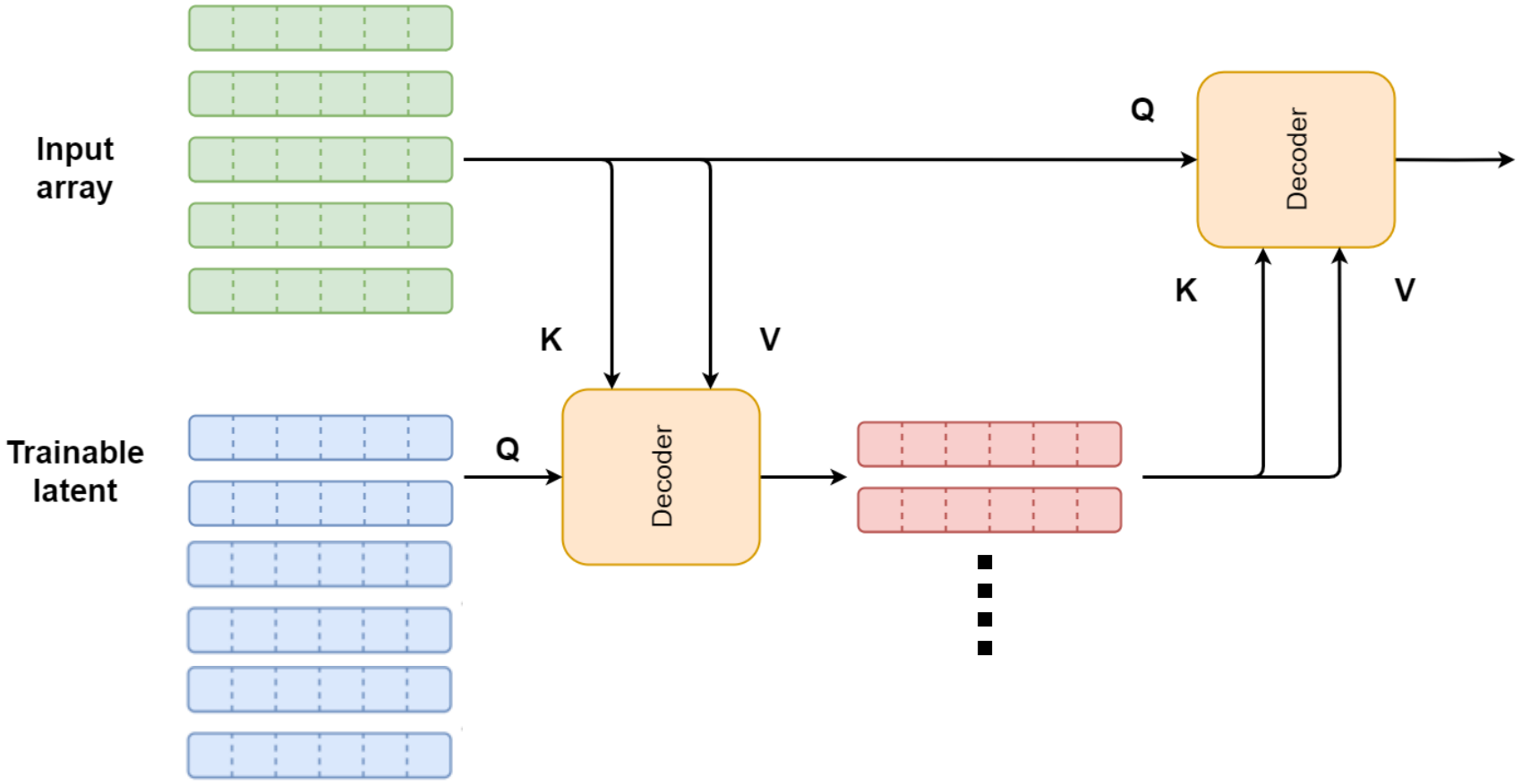
Lets said we have a pc of ~2000 points and want to exchange information between points:

- SA is expensive (2000**2)
- CA would be (2000*m)
  - But what can m be?
- Trainable point cloud
  - w/ active gradient



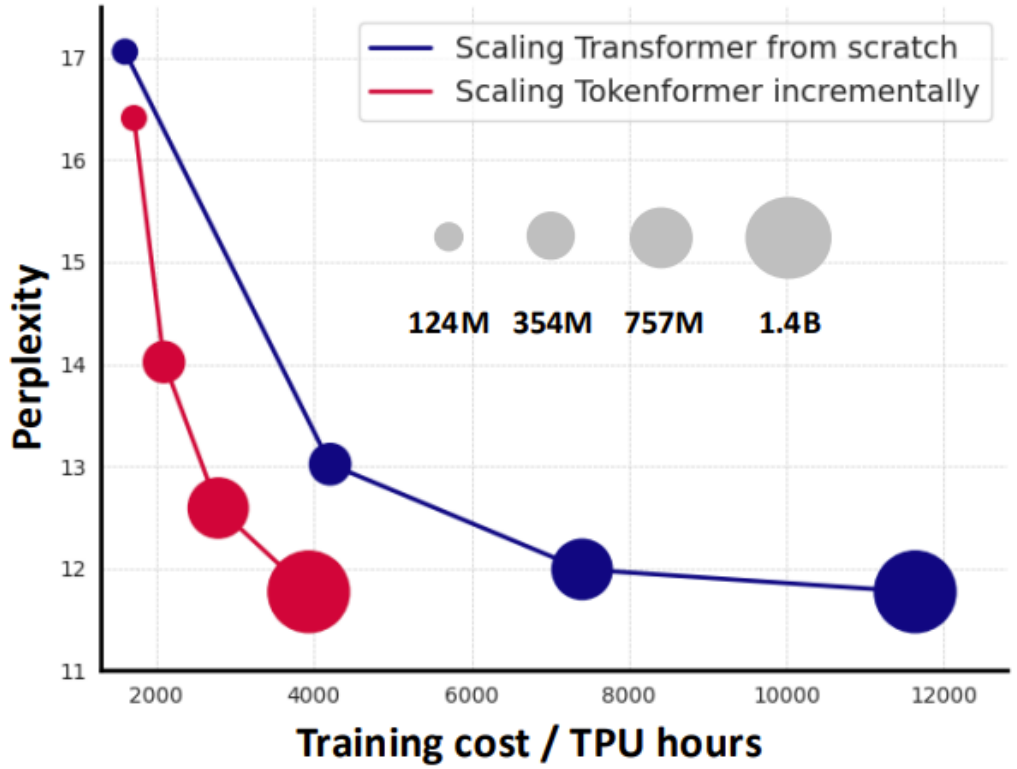
Transformers: Tokenformers

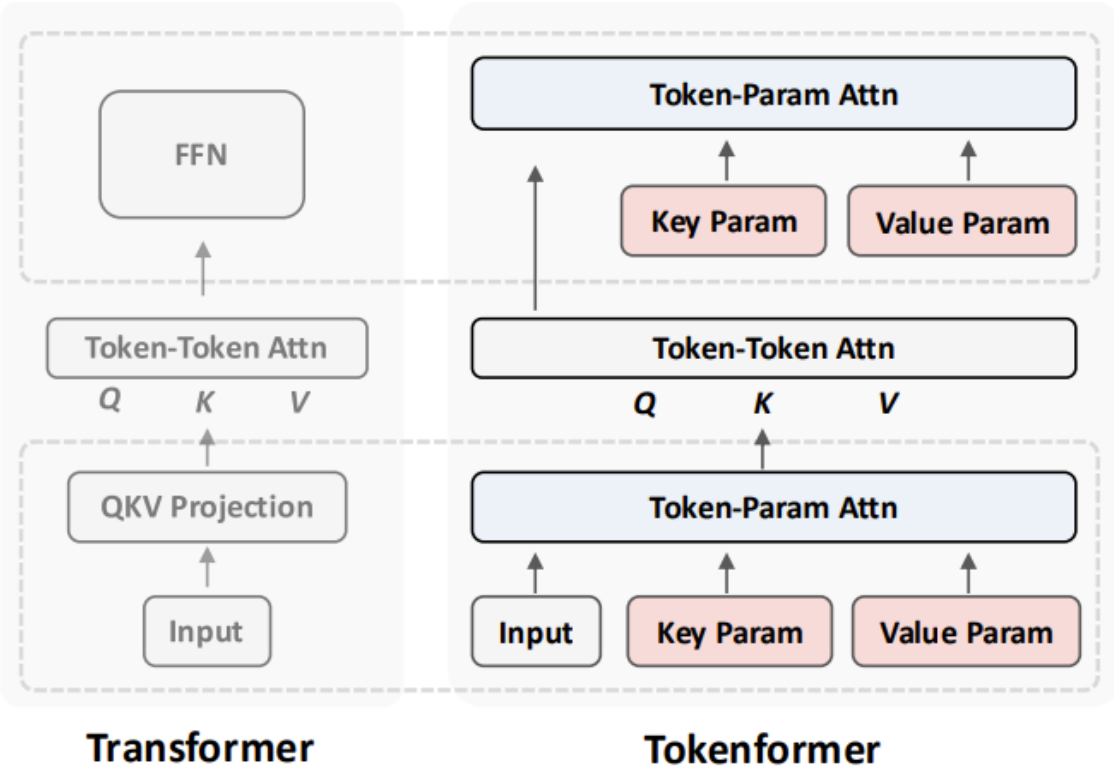
TokenFormer: Rethinking Transformer Scaling with Tokenized Model Parameters



Transformers: Tokenformers

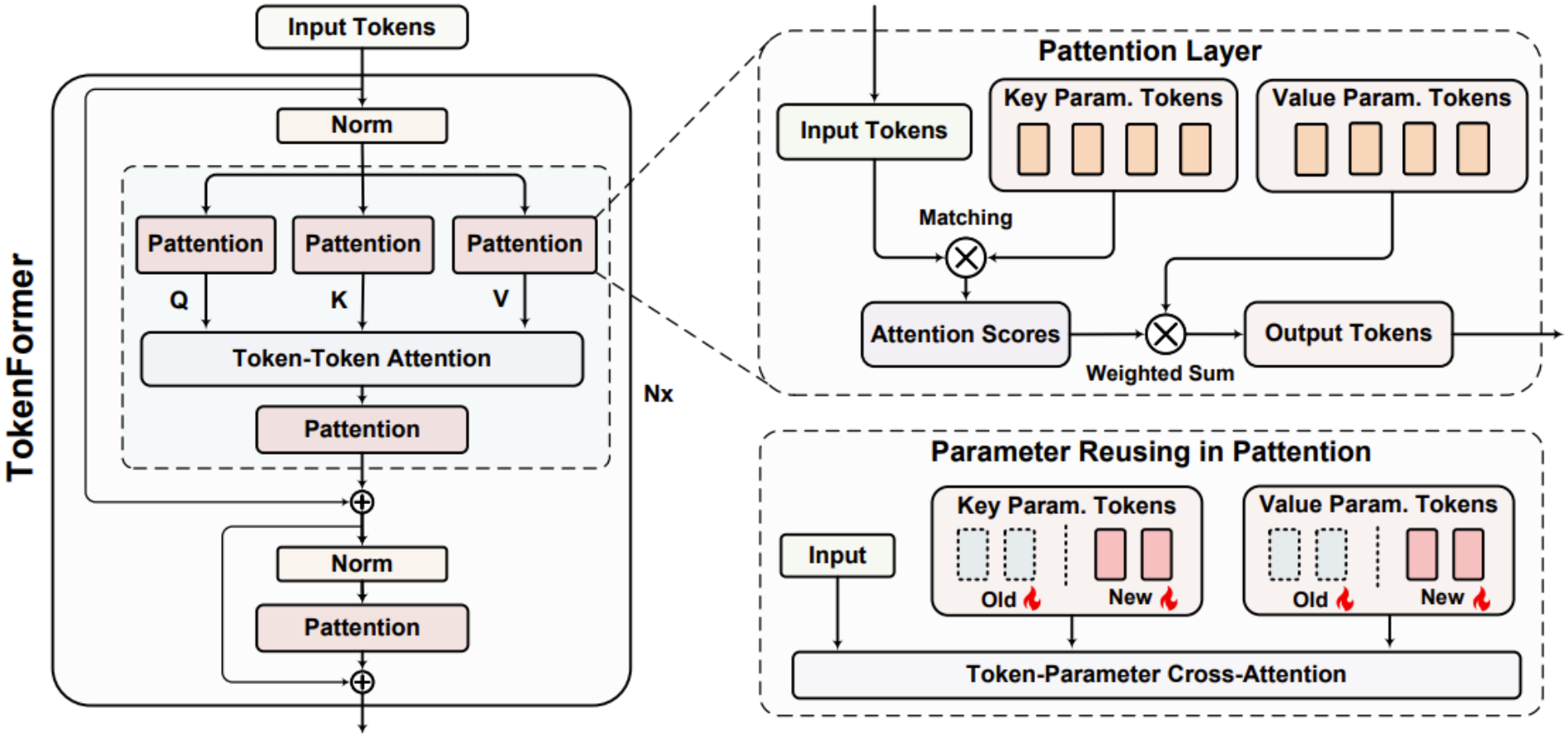
TokenFormer: Rethinking Transformer Scaling with Tokenized Model Parameters





Transformers: Tokenformers

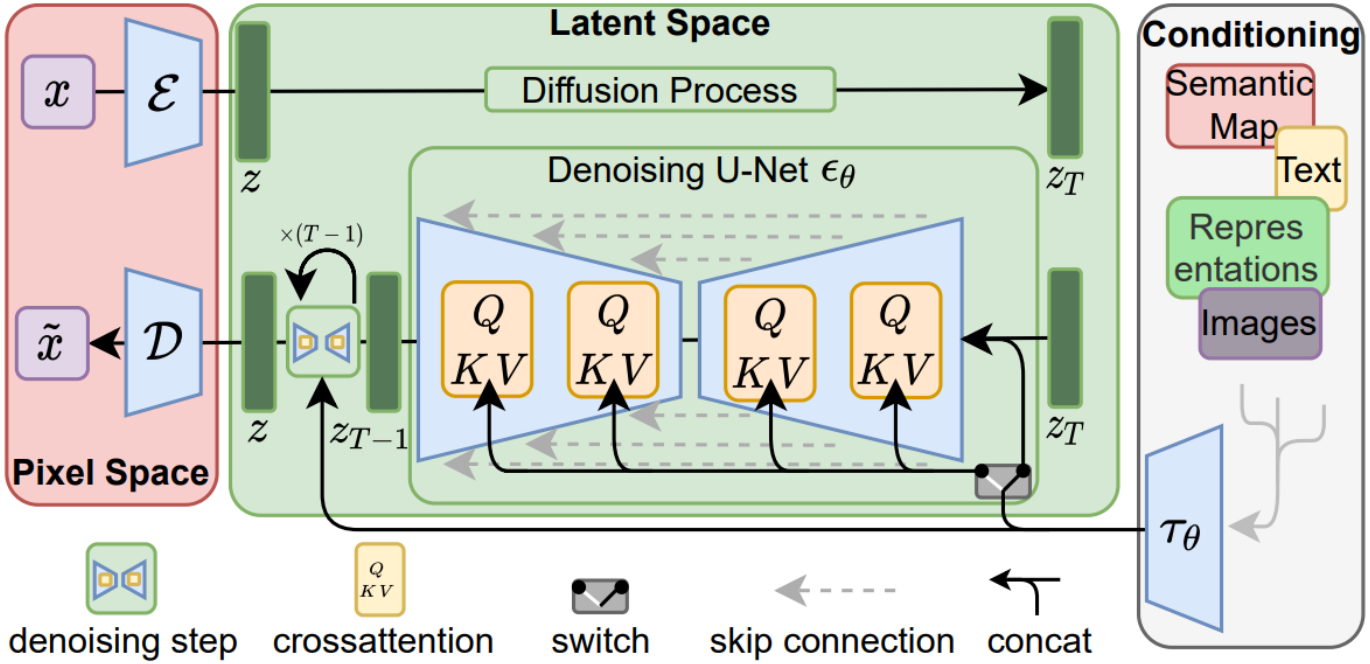
TokenFormer: Rethinking Transformer Scaling with Tokenized Model Parameters

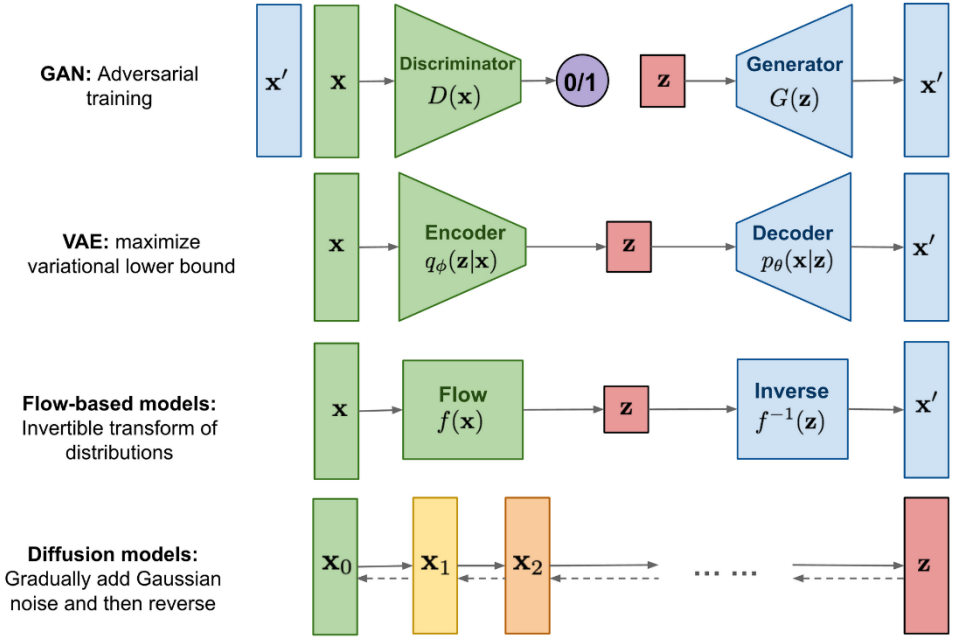


VAE are not dead!

High-Resolution Image Synthesis with Latent Diffusion

- Use a (V)AE to compress image
- Run diffusion in the latent space





Overview

- Variational inference is taking over (Conditional generation is king)
  - Using log likelihood
  - Normalising Flow and Diffusion - Probabilistic regression
  - Multi-task learners
- Modern transformer architecture on a compute budget
  - Mixture-of-Depths
  - Perceiver IO
- “The Death of Transformers” – The Rise of Tokenformers!!!
- Optimal Transport (if time allows it)